Manuale di utilizzo del software Codesys

L'ambiente di sviluppo di Codesys è un marchio della 3S-Software, il seguente non è un manuale ufficiale, ma è tratto dal manuale in lingua inglese della 3S-Software ed alcune applicazioni svolte all'interno dell'università di Bologna. Nel caso si trovassero errori potete comunicarli a Matteo Sartini, matteo.sartini@unibo.it (versione 1.1)

Contents

1 Introduzione

L'ambiente di sviluppo Codesys (Controlled Development System) sviluppato dalla ditta 3S Software (www.3s-software.com) permette di implementare automi per il controllo logico utilizzando uno dei linguaggi definiti dalla normativa IEC 61131-3. La demo dell'ambiente di sviluppo è scaricabile dal sito, ha durata illimitata e presenta tutte le componenti della versione completa, le uniche limitazioni sono per le librerie fornite e per il numero massimo dei componenti da poter utilizzare, che nei linguaggi SFC e ST non ha praticamente limitazioni, mentre nei linguaggi LD ed altri sono limitati nel numero di righe da poter utilizzare.
Il tool di sviluppo viene utilizzato per due tipologie di applicazioni:

- Per programmare ed eseguire su PC controllori per macchine automatiche, per queste applicazioni Codesys funziona come compilatore di programmi a norma IEC 61131-3, generando un programma eseguibile per l'ambiente di esecuzione real time CoDeSys SP RTE.
- Per programmare sistemi di controllo industriali (come i PLC), per queste applicazioni il costruttore di PLC acquista il programma Codesys con integrato il compilatore e una interfaccia personalizzata per quello specifico PLC.

Una volta generato il codice di controllo, Codesys integra nel suo ambiente una modalità di funzionamento chiamata Simulation mode, che permette di eseguire il programma sviluppato senza avere ne l'ambiente real time, ne un vero e proprio PLC. E' possibile creare una propria interfaccia grafica per interagire con il codice di controllo, a questa interfaccia è possibile collegare un programma che il compito di "simulare" il comportamento del plant per il quale è stato scritto il codice di controllo. In questo modo mandando in esecuzione entrambi i programmi (codice di controllo, codice per simulare il plant) è possibile "testare" il codice di controllo; inoltre tramite l'interfaccia grafica è possibile interagire durante la simulazione. Nella versione demo del tool è possibile lavorare soltanto in simulation mode, cioè non è possibile generare codice per l'ambiente real time o per PLC.

2 Definizioni e richiami della norma IEC61131-3

Lo standard IEC 61131-3 è la terza parte di una norma più generale (IEC 61131) che fornisce una standardizzazione dei PLC:

IEC 61131-1 Definizione del dispositivo PLC.

IEC 61131-2 Architettura hardware e software dei PLC.

IEC 61131-3 Linguaggi di programmazione per implementare i controlli di sequenze sui PLC.

IEC 61131-4 Linee guida per l'utente.

IEC 61131-5 Descrizione dei servizi di messaggistica.

IEC 61131-6 Comunicazione tramite bus di campo (fieldbus).

IEC 61131-7 Programmazione di controllo a logica sfumata (fuzzy logic).

IEC 61131-8 Linee guida per l'applicazione e l'implementazione di linguaggi di programmazione.

Lo standard IEC 1131-3 fornisce una definizione dei linguaggi di programmazione per i controllori a logica programmabile, allo scopo di far evolvere verso una normalizzazione di tali linguaggi in modo che il programmatore possa astrarre il più possibile dall'architettura del particolare controllore che verrà utilizzato. Per la medesima ragione si sono definite una serie di caratteristiche comuni a tutti i linguaggi, relative in particolar modo alla sintassi di dichiarazione di variabili simboliche e di moduli software.

2.1 Il modello software

L'IEC ha elaborato un modello del sistema di controllo, cercando di definirne i suoi componenti software. Per fare ciò sono state considerate tutte le interazioni fra il programma e l'ambiente operativo del sistema di controllo. Alla fine si è optato per un modello software di tipo stratificato e gerarchico, che si basa su di un insieme ben preciso di elementi (oggetti), ognuno dei quali si trova ad un determinato livello della gerarchia e racchiude gli elementi del livello sottostante. Tale gerarchia, può essere schematicamente descritta dalla figura 1. I componenti sono quindi distribuiti secondo una struttura gerarchica simile ad un sistema ad oggetti in cui ogni elemento ha caratteristiche proprie ed è connesso agli altri secondo opportuni vincoli di aggregazione, appartenenza o utilizzo.
Segue la presentazione dei vari componenti del modello:

Configuration: E' l'elemento che contiene tutti gli altri elementi, e corrisponde all'intero sistema controllore programmabile, ovvero, generalmente ad un PLC. In un sistema di controllo è possibile avere più configuration, le quali possono comunicare tra loro (le modalità di comunicazione sono descritte nella parte quinta della norma).

Resource: E' l'oggetto che rappresenta il supporto di esecuzione dei programmi. Di conseguenza un programma IEC per poter funzionare deve necessariamente esere caricato in una resource.
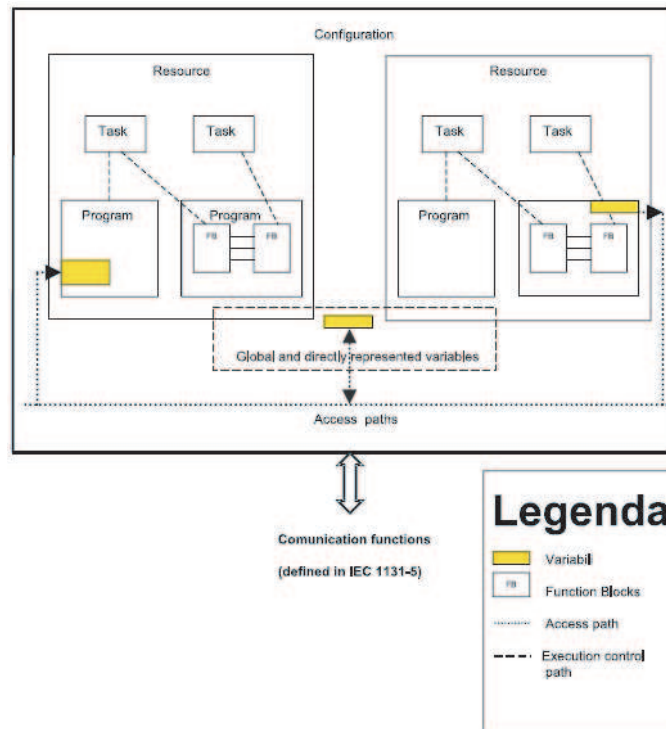
Figure 1: *Modello Software*

Lo standard definisce una Resource come: "Un elemento che corrisponde ad una unità funzionale di elaborazione dei segnali, connessa a dispositivi di interfaccia uomo-macchina e con funzioni di interazione con sensori ed attuatori". Quindi, una delle principali funzioni di questa componente del modello software è di fornire un'interfaccia fra il programma e gli I/O fisici del PLC, oltre che con l'uomo (tramite i terminali di programmazione). Le Resource sono componenti del modello software tra loro autonome e sono, solitamente, poste in corrispondenza biunivoca con un diverso processore (CPU). Queste componenti possono aver accesso alle Global Variables ed alle Directly Represented Variables (v. seguito) essendo definite a livello di Configuration, le quali, quindi, costituiscono un meccanismo di comunicazione fra le Resource.

Program: Lo standard IEC 1131-3 lo definisce: "L'insieme logico di tutti gli elementi di programmazione ed i costrutti necessari per la funzionalità di elaborazione di segnali richiesta ad un sistema controllore programmabile." I Program si trovano necessariamente all'interno di una Resource. Essi contengono dei costrutti realmente eseguibili, programmati nei linguaggi previsti dallo standard. L'esecuzione di un Program avviene sotto il controllo di uno o più Task (v. seguito).

Task: E' l'elemento che si occupa di controllare un intero program o una parte di esso. Ogni task può essere attivato o periodicamente o al verificarsi di un determinato evento.

Local and Global variables: Gli identificativi delle variabili sono dei nomi simbolici a cui il programmatore attribuisce un determinato significato, dipendente dal tipo di dato e

dalla zona di memoria incui si richiede sia allocata. Possono essere definite come locali, quindi accessibili unicamente dalle unità nelle quali sono dichiarate (Programs e moduli di programma, Resources, Configurations); o come globali. Nel secondo caso risultano accessibili nell'unità che le ha dichiarate ed in tutti gli oggetti in essa contenuti secondo la struttura gerarchica definita nel modello software.

Directly represented variable: Sono variabili che si riferiscono a indirizzi di memoria del PLC. Lo Standard suggerisce uso di queste variabili limitato al livello di Program, in modo da mantenere un certo grado di riutilizzabilità del codice.

Function Blocks: Sono moduli di programma riutilizzabili, grazie al quale è possibile ottenere programmi strutturati. Un Function Block è solitamente formato da due componenti:

- DATI: insieme delle variabili utilizzate all'interno del Function Block (parametri in ingresso e in uscita, variabili interne).
- ALGORITMO: insieme di istruzioni che costituiscono il corpo del Function Block, vengono eseguite ogni volta che viene chiamato.

L'algoritmo elabora un nouvo set di paramentri in uscita ed eventualmente un nuovo valore delle variabili interne a partire dal valore corrente degli ingressi e delle stesse variabili interne. Le variabili manipolate all'interno di un Function Block possono essere locali o globali. L'IEC 61131-3 definisce una insieme di Function Block standard (tra i quali temporizzatori e contatori).

Functions: Le Functions sono anch'esse moduli di programma riutilizzabili, che ricevono in ingresso un set di valori, e producono in uscita un solo valore. Nei Function Blocks il valore delle variabili interne veniva mantenuto tra due esecuzioni successive mentre le variabili interne di una Function possono essere solo temporanee. In sostanza, mentre i Function Blocks sono dotati di un proprio stato interno, le Function mancano di tale proprietà, per cui, in corrispondenza della stessa configurazione d'ingresso esse forniscono sempre lo stesso risultato. Anche per questi oggetti l'IEC specifica una serie di funzioni standard.

Access path: E l'ultimo degli elementi del modello software introdotto dallo standard. Poiché un sistema di controllo può comporsi di diverse Configuration, le quali devono scambiarsi dati ed informazioni attraverso collegamenti di tipo remoto, è necessario supportare tale trasferimento con opportuni protocolli. Sono quindi dei percorsi di comunicazione fra le varie Configuration ai quali è possibile accedere attraverso un set di speciali variabili, dichiarate usando il costrutto `VAR_ACCESS`. Questo set di variabili funge da interfaccia per la Configuration in cui viene dichiarato, nel suo relazionarsi con eventali altre Configuration remote.

2.2 Program Organization Units

Lo standard IEC definisce come Program Organisation Units (POU) i seguenti tre elementi del modello software: Programs, Function blocks e Functions. Questi tre oggetti sono gli elementi più importanti definiti dalla norma in quanto definiscono finalmente una modalità di organizzazione

della struttura software efficace ed orientata al riuso. La proprietà che hanno in comune questi elementi è che ognuno di essi può essere replicato in diverse parti di una applicazione, qualora le sue funzionalità debbano essere ripetute. Quando si definisce una POU associandole un nome (generico) che identifica un tipo di quella particolare POU, ancora in realtà non si è generato un vero e proprio modulo di programma eseguibile. Infatti per poter inserire una POU (ad eccezione delle Function) in un applicativo essa va **istanziata** cioè dichiarata tra i moduli effettivamente utilizzati assegnandole un nome simbolico specifico ed univoco. In questo modo è possibile definire e utilizzare più istanze di una stessa POU in un unico applicativo. Una POU può, in linea di principio, essere programmata in uno qualunque dei linguaggi delle norma descritti nel seguito e contenere chiamate a qualunque altra POU istanziata. Lo standard vieta in modo esplicito solamente l'uso ricorsivo di POU: tale scelta è motivata dall'esigenza di determinismo delle prestazioni real-time dell'applicativo.

2.3 I linguaggi di programmazione

Un aspetto dei controllori industriali tradizionalmente ostile per il progettista è la difformità dei linguaggi di programmazione. Per questo la norma ha inteso regolalamentare questo aspetto definendo cinque linguaggi standard dalle caratteristiche differenti, in grado di coprire tutte le necessità nello sviluppo di un programma di controllo. I linguaggi definiti nella norma IEC 61131-3 sono:

- **Instruction List (IL)**: Linguaggio testuale di basso livello, altamente utilizzato per piccole applicazioni o per ottimizare parti di un'applicazione.
- **Sequential Function Chart (SFC)**: E' un linguaggio grafico estremamente espressivo. E' molto usato in quanto permette di partizionare in una sequenza di stati consecutivi l'esecuzione dell'applicativo.
- **Structured Text (ST)**: E' un linguaggio testuale strutturato di alto livello, simile al Pascal e al C, creato appositamente per la programmazione automatizzata di processi. Viene generalmente usato per implementare complesse procedure non facilmente esprimibili attraverso linguaggi grafici.
- **Ladder Diagram (LD)**: Linguaggio grafico che implementa una logica booleana servendosi di schemi elettrici.
- **Function Block Diagram (FBD)** E' un linguaggio grafico, basato sull'interpretazione del comportamento del sistema in termini di flusso dei segnali tra gli elementi del processo (analogia con il diagramma dei flussi di segnali all'interno di un circuito elettronico).

Per la descrizione dell'analisi strutturale dei linguaggi introdotti dalla norma si rimanda alla consultazione di testi specializzati che hanno come argomento la divulgazione della stessa.

3 L'ambiente CoDeSys: Struttura e programmazione

Una volta installato il tool di sviluppo si può vedere come esso presenti più programmi al suo interno, il principale è **Codesys V2.3** all'interno del quale è possibile sviluppare e simulare il codice per il controllo di sistemi industriali secondo la norma IEC 61131-3. Il pacchetto comprende anche:

- il programma **Configuration** che serve per configurare i server OPC e DDE. Codesys supporta questi due server per la comunicazione e le applicazioni remote.
- l'interfaccia **ENI** che permette di connettere l'ambiente di programmazione CoDeSys ad un database esterno, in modo che i dati possano essere eventualmente condivisi da più user, progetti o programmi.
- il programma **CoDeSys SP RTE** utilizzabile per esecuzioni realtime (non utilizzabile nella versione demo).
- il sistema runtime **HMI** per le esecuzioni di visualizzazioni grafiche create in ambiente CoDeSys (non utilizzabile nella vesione demo).

Nel tool di sviluppo con il termine **Progetto** si intendono tutti gli oggetti necessari per la stesura del programma per PLC, gli oggetti sono: POU (Program Organization Unit), tipi di dato definiti dall'utente, la parte di visualizzazione, risorse e librerie. Ciascuna **POU** consiste di una parte di dichiarazione e da una parte centrale scritta secondo uno dei linguaggi previsti dalla norma IEC, per poter utilizzare questi linguaggi è però necessario caricare la libreria `standard.lib` nel progetto; una POU può chiamare un altra POU ma non è ammessa ricorsione.

3.1 Primi passi

Lanciando il programma CoDeSys, esso carica l'ultimo progetto su cui si è lavorato, o se il programma viene lanciato per la prima volta carica un esempio. Per creare un nuovo progetto basta cliccare su `File` ⟶ `New` e quando appare **Target setting** selezionare `none`, a questo punto appare la schermata come quella di figura 2 che indica il nome da dare al programma, ed in che linguaggio si vuole scriverlo. Di default la nuova POU è un **Program** in linguaggio ST con nome PLC_PRG, se non verranno cambiate le impostazioni nella configurazione del progetto nella **Task configuration** l'unica POU eseguita automaticamente dal sistema sarà proprio il Program col nome PLC_PRG, tutte le altre POU saranno eseguite solo se richiamate esplicitamente.
Una volta creato un nuovo progetto oppure, aperto un progetto esistente cliccando su `File` ⟶ `Open` CoDeSys si presenta con la schermata come quella di figura 3. La schermata è divisa in due parti, nella parte destra viene visualizzata la finestra collegata all'elemento selezionato nella parte sinistra; nella parte sinistra viene visualizzato il **Browser di progetto**, costituito da quattro sotto schermate che sono selezionabili cliccandoci sopra nella parte in basso a sinistra. I quattro oggetti che compongono il browser di progetto sono:

- POU: elenco delle unità di programma (Program Organization Unit), distinte come specificato dalla norma IEC 61131-3 in Program (PRG), Function(FC) e Function Blocks (FB).
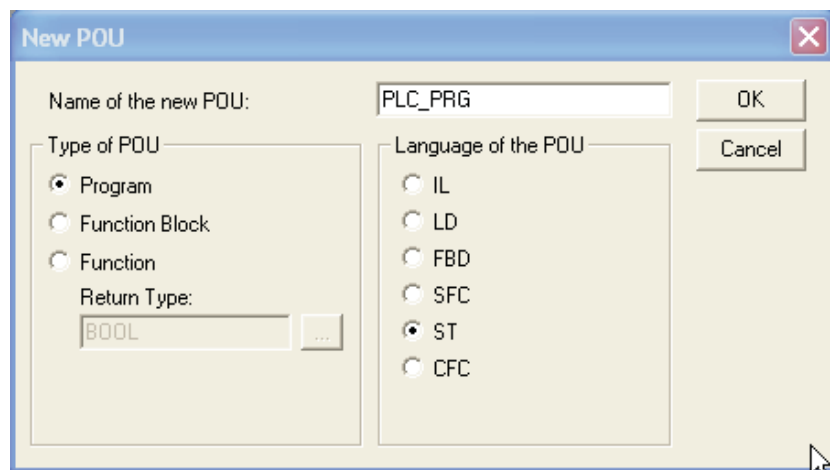
Figure 2: *Nuova Pou*
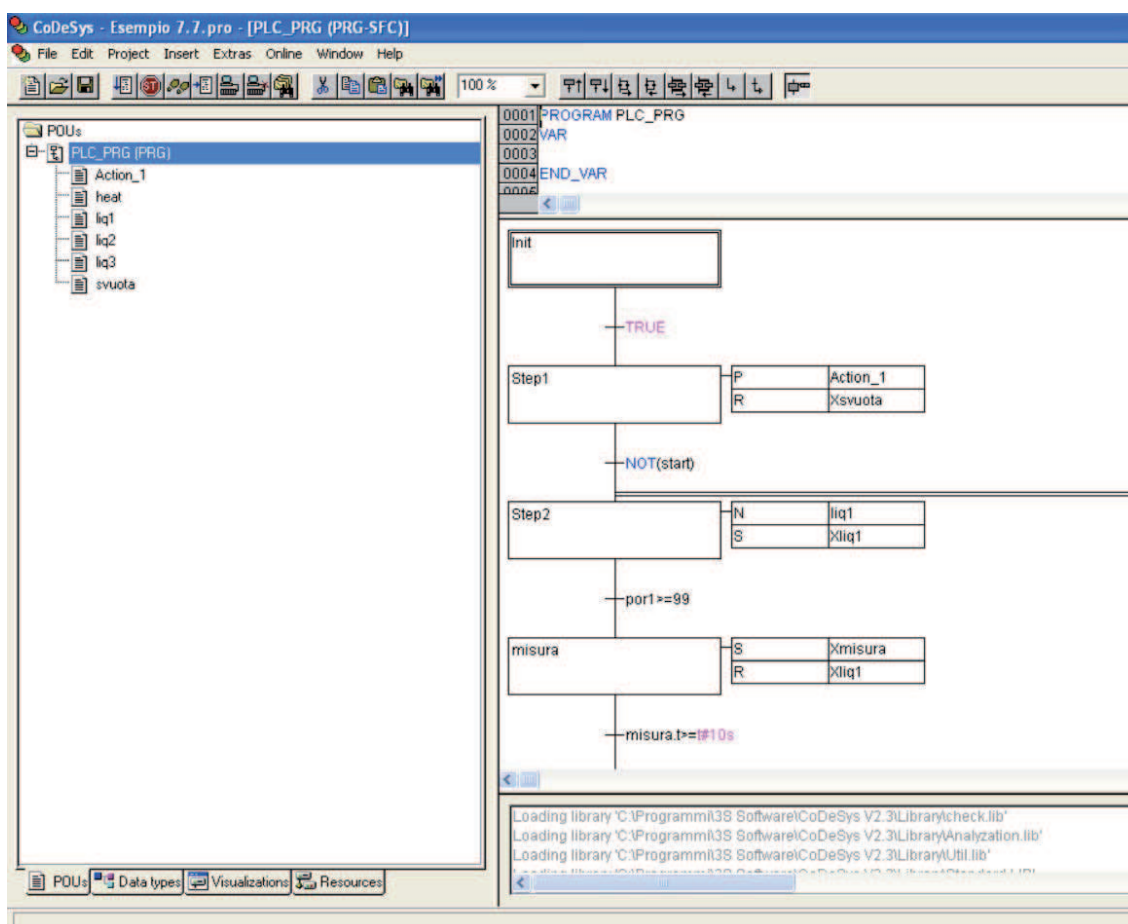
Figure 3: *Browser di progetto*

- Data types: elenco dei tipi di dato definiti dall'utente (es strutture dati, tipi enumerativi, ecc.)
- Visualizations: elenco dei pannelli grafici di visualizzazione.
- Resource: elenco dei menù di configurazione degli elementi comuni del progetto (es variabili globali, tasks di esecuzione, librerie, configurazione del sistema, PLC Target, ecc.)
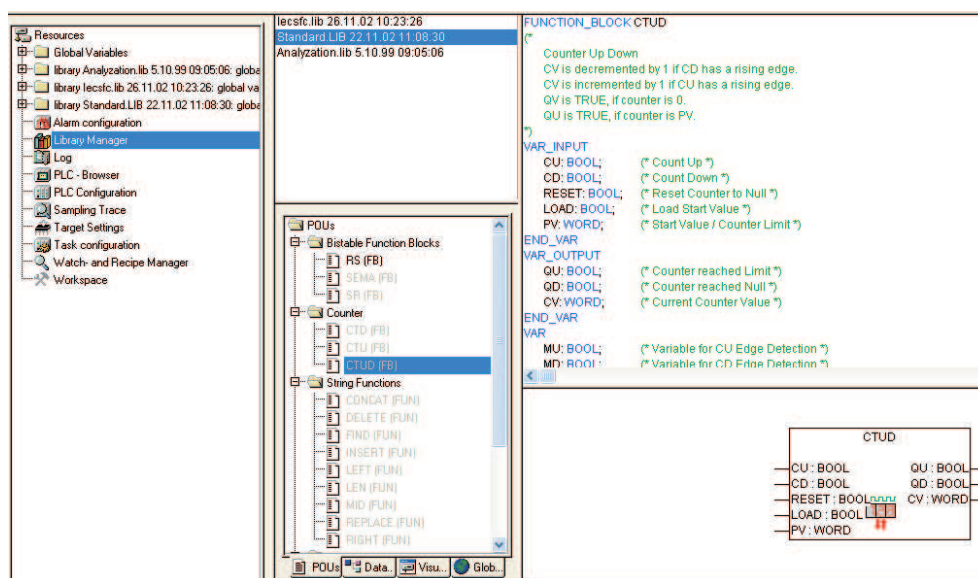
Figure 4: *Browser di libreria*

In qualsiasi momento è possibile selezionare una delle quattro finestre che compongono il browser di progetto. In POU è presente tutto l'elenco dei programmi presenti nel progetto, se si vuole aggiungere una nuova unità basta cliccare su POUs e selezionare `Add object`, così facendo appare la finestra come quella di figura 2 ed è possibile selezionare in che linguaggio scrivere il nuovo programma. CoDeSys offre anche la possibilità di convertire un programma scritto in un linguaggio in un altro linguaggio, questo è possibile cliccando sulla POU e selezionando `Convert object`, apparirà una finestra dove si selezionerà il linguaggio in cui convertire il programma ed il nuovo nome; la POU PLC_PRG una volta definito il linguaggio non può essere convertita.
Per utilizzare i linguaggi standard per scrivere i programmi, bisogna caricare le librerie che supportano questi linguaggi, per inserirle bisogna andare nella finestra **Resource** e selezionare cliccando due volte **Library Manager**. Nella parte destra compare una schermata simile alla schermata principale di CoDeSys, ma che mostra l'elenco delle POU programmate che si possono inserire nel progetto (timer, contatori, registri, ecc...) mostrando sia la dichiarazione delle variabili sia l'aspetto grafico dell'oggetto visibile quando si inserisce durante la programmazione nei linguaggi LD e FBD. Per inserire delle nuove librerie bisogna cliccare col tasto destro nella sotto finestra in alto dove è presente l'elenco delle librerie, e selezionando `Additional Library` è possibile inserire ulteriori librerie nel progetto scegliendo un file con estensione **.LIB**.
Il progetto per lo sviluppo di un sistema di controllo all'interno di CoDeSys può essere pensato diviso in due programmi principali, uno è il programma che si occupa del controllo logico del

sistema che si vuole controllare e lo chiameremo controllo, mentre l'altro programma è quello che simula l'impianto, il plant del progetto e lo chiameremo simulatore. Solitamente il programma controllo viene scritto in linguaggio SFC, mentre il programma simulatore viene scritto in linguaggio strutturato, naturalmente questi due programmi devono interagire tra loro, dal punto di vista degli ingressi e delle uscite di un PLC, il programma controllo genera le uscita una volta elaborati gli ingressi, cioè deve comandare attraverso le azioni gli attuatori del PLC. Il programma
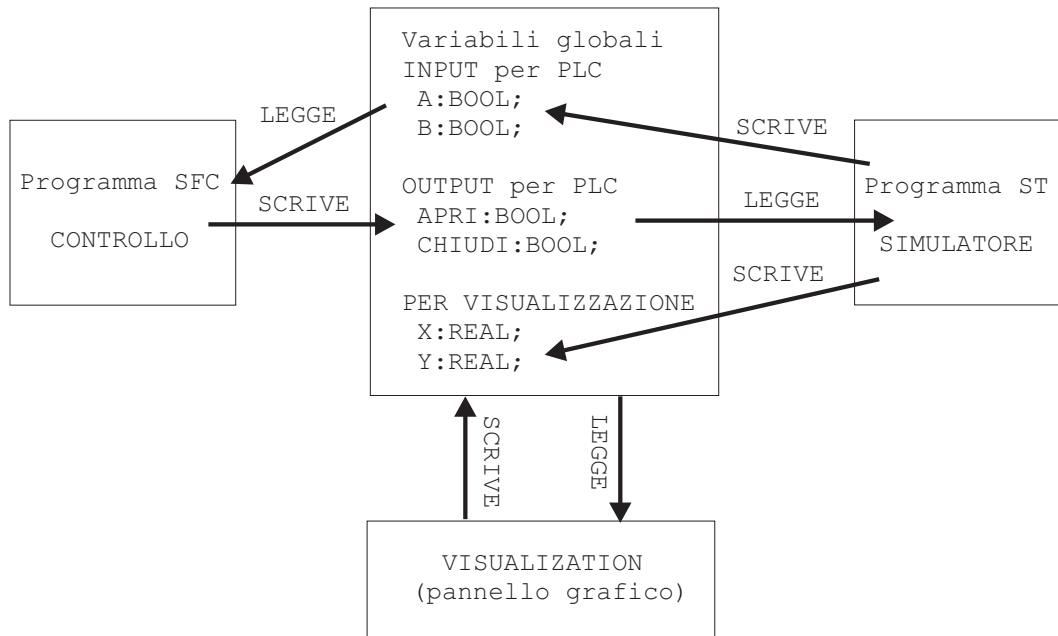
Figure 5: *Schema funzionamento di Codesys*

simulatore invece legge lo stato dei comandi impartiti tramite SFC agli attuatori, e modifica lo stato dei sensori installati, che sono gli ingressi del PLC, e modifica il valore di opportune variabili che vengono utilizzate per la visualizzazione sul pannello grafico, in maniera coerente con il funzionamento dell'impianto. Se ad esempio SFC comanda il movimento di un carroponte verso l'alto, il simulatore dovrà decrementare il valore di una variabile chiamata ad esempio Y, ed imporre un valore TRUE al sensore di fine corsa quando la variabile Y arriva ad un certo valore limite. I due programmi sviluppati dovranno interagire tra loro, come mostrato nella figura 5, quindi le variabili dovranno essere definite globali, le uniche variabili locali saranno quelle che vengono utilizzate soltanto all'interno di un programma e che non devono leggere o scrivere sugli ingressi/uscite del sistema. Entrambi i programmi devono essere eseguite, l'unico però che viene eseguito di default è quello chiamato PLC_PRG, bisogna quindi configurare il sistema in modo che esegua entrambi i programmi. Per fare questo bisogna modificare il progetto andando a selezionare la finestra **Resources** e selezionando l'opzione `Task configuration`, cliccandoci sopra due volte, e poi cliccare col tasto destro su `Task configuration` quello nella parte destra dello schermo, non sul browser di progetto. Una volta selezionato il comando `Append Task` e definito un nuovo task, chiamato ad esempio Main come nella figura 6, bisogna configurare gli attributi del task ed inserire da qui tutti i programmi che si vogliano eseguire su quel task attraverso il comando `Append program call`. Nell'esempio vengono chiamati i due programmi PLC_PRG e simulazione.

Il tool di sviluppo permette di simulare il progetto una volta completato in tutte le sue parti,
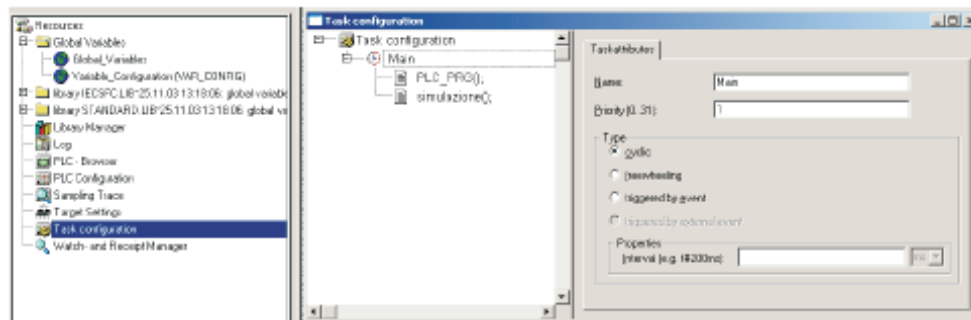
Figure 6: *Finestra di configurazione dei task*

per lavorare in modalità simulazione bisogna che sia impostata l'opzione `Simulation Mode`, che si trova nel menù **Online**. Se quando si è creato il progetto si era impostato il target del progetto come none, allora l'opzione Simulation mode dovrebbe essere già impostata e non modificabile. Una volta terminata la programmazione di tutti i blocchi e definite le variabili, è possibile verificare la correttezza del progetto attraverso il comando `Build` che si trova nel menù **Project** (oppure premendo il tasto F11 della tastiera del PC), nel caso in cui il programma non sia corretto nella parte inferiore della finestra viene visualizzato l'errore; cliccando due volte sopra l'errore, viene aperta automaticamente il programma nel punto dell'errore. Se la compilazione non riporta errori, allora è possibile eseguire il progetto in modalità simulazione, scegliendo dal menù **Online** il comando `Login` (oppure usando la combinazione dei tasti ALT e F8) e poi scegliendo sempre dallo stesso menù il comando `Run` (oppure usando la combinazione dei tasti ALT e F5), in questo modo il programma è in esecuzione sul PC, il quale simula il funzionamento del PLC.

3.2 L'Editor SFC

L'SFC è il linguaggio grafico che ci permette di definire l'ordine cronologico delle differenti azioni all'interno di un programma. Come già detto, di solito il programma che si occupa del controllo logico del progetto (il PLC_PRG) è scritto in SFC. Questo tipo di linguaggio grafico consiste in una serie di STEP che sono connessi l'un l'altro attraverso connessioni dirette (transitions). Creando un nuovo file ed aprendo il Program PLC_PRG la schermata di CoDeSys appare come quella in figura 7. Possiamo vedere che lo step iniziale di ogni programma in SFC viene chiamato **Init** ed è contraddistinto da un blocco con il bordo doppio.
L'Editor SFC in ambiente CoDeSys mette a disposizione due tipi di STEP:

- Lo **Step semplice** permette di definire al suo interno le azioni che devono essere eseguite in quel determinato blocco. Se uno step è stato implementato (cioè se al suo interno è stata definita un'azione in un determinato linguaggio) viene visualizzato un piccolo triangolo nero nell'angolo in alto a destra. In modalità on line, quando si sta eseguendo lo step (ovvero quando lo step è attivo) vengono eseguite tutte le istruzioni definite al suo interno fino a che la transizione che lo lega al blocco successivo non diventa TRUE.

- **L'IEC-Step** è un particolare tipo di step in cui le azioni da eseguire vengono definite separatamente in base alla loro durata all'interno del tempo di esecuzione del programma.
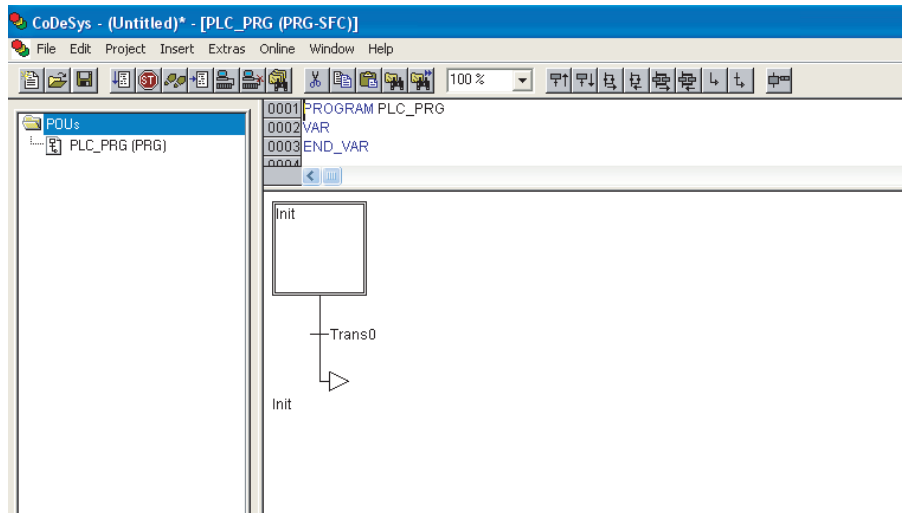
Figure 7: *Nuovo programma in SFC*

L'utilizzo di questi step è subordinato all'inclusione nel progetto della libreria **Iecsfc.lib**. Inoltre ogni volta che si vuole inserire uno step di questo tipo è necessario settare l'opzione **'Extras'Use IEC-Steps'**. Se poi si vuole che il blocco Init sia IEC-Step sarà necessario
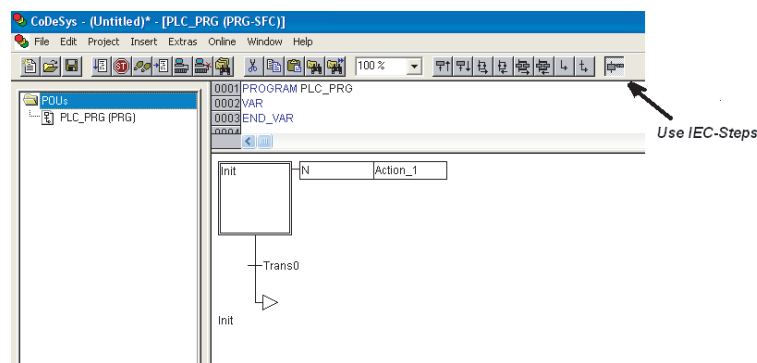
Figure 8: *Nuovo programma (Il blocco iniziale è un step IEC)*

cancellare il PLC-PRG e riaprirlo dopo aver selezionato l'opzione sopraindicata. (vedi figura 8).

Le azioni associate ad un IEC-STEP vengono visualizzate alla destra dello step stesso attarverso un box diviso in due parti. La parte sinistra contiene il qualificatore che ne definisce la durata (vedi tabella Lista qualificatori) mentre il campo destro contiene il nome dell'azione o il nome della variabile da settare a TRUE.

Le azioni da associare ad un IEC-Step devono essere 'appese' direttamente sotto il programma in cui vengono usate. Questo è possibile cliccando con il tasto destro sul nome del programma presente nell'editor di progetto, e scegliendo poi nella finestra a discesa il comando `Add Action` (vedi figura 10). Non è possibile inserire direttamente delle azioni in un IEC-Step. Se volessi
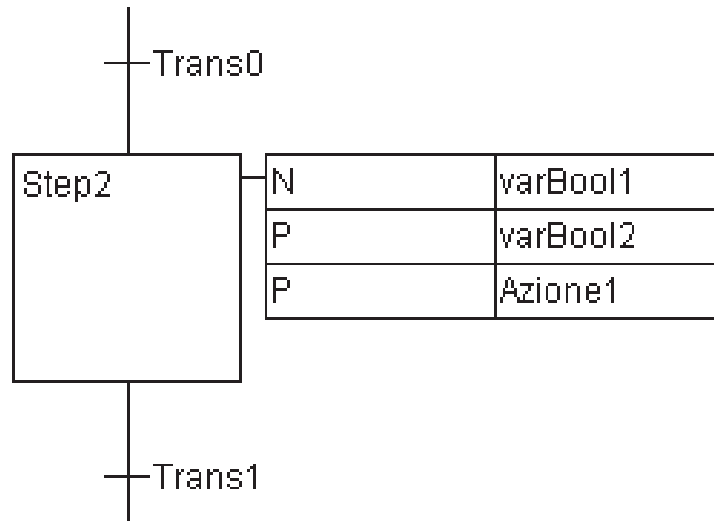
Figure 9: *IEC-STEP e azioni associate*

N	L' azione viene eseguita per tutta la durata dello step
S	L' azione viene eseguita e rimane attiva fino ad eventuale azione di Reset (R)
R	L' azione eseguita con SET viene disattivata
L	L' azione eseguita termina dopo un certo intervallo di tempo, che può durare al massimo fino a che lo step è attivo
D	L' azione viene eseguita dopo un certo tempo e poi rimane attiva per tutta la durata dello step
P	L' azione viene eseguita solo una volta se lo step è attivo
SD	L' azione viene eseguita come azione SET dopo un certo intervallo di tempo dall'attivazione dello stato e rimane attivo fino ad un Reset (R)
DS	L' azione viene eseguita come azione SET se lo stato rimane attivo per un certo intervallo di tempo e rimane attiva fino ad un Reset (R)
SL	L' azione viene eseguita come SET, dopo un certo tempo l'azione l'azione viene terminata con un' azione RESET

Table 1: *Lista dei qualificatori*
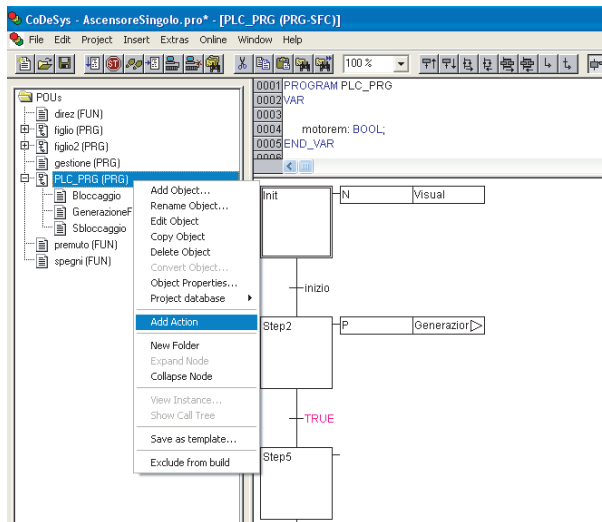
Figure 10: *Creare un azione utilizzabile da un Iec-Step*

aggiungere due azioni *Action_1* e *Action_2* queste devono essere definite a parte e solo richiamate con il nome all'interno del rettangolo a fianco dello step. Per definire una nuova azione associata ad una POU basta cliccare con il tasto destro del mouse sulla POU e selezionare *Add Action*. A questo punto si apre una finestra dove ci viene chiesto il linguaggio in cui verrà scritta la nuova azione e il nome che le vogliamo associare. In base al linguaggio scelto si aprirà il rispettivo editor per la scrittura dell'azione.
L'Editor SFC di CoDeSys permette inoltre di definire delle azioni addizionali per ogni tipo di step che vengono eseguite una sola volta o non appena lo step diventa attivo (**Entry Action**) o un attimo prima di disattivarsi (**Exit Action**). Uno step contenente una "entry action" è indicato con una "E" nell'angolo in basso a sinistra, mentre una "exit action" è contrassegnata da una "X" nell'angolo in basso a destra.
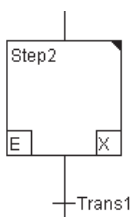
Figure 11: *Step che contiene un Azione normale, una entry action, ed una exit action.*

Variabili implicite: Le variabili implicite sono delle variabili che immagazzinano lo stato di uno step. Vengono spesso utilizzate per regolare o controllare il flusso di esecuzione del programma SFC e sono perciò visibili e utilizzabili negli altri programmi del progetto. Esiste una variabile implicita per ogni step, che è identificata con <**nomedellostep.x**>

per gli step IEC o <**nomedellostep**> per gli step semplici. Quando lo step è attivo assumono il valore TRUE e viceversa sono FALSE se lo step è inattivo. E' possibile inoltre utilizzare la variabile <**nomedell'azione**>**.x** che permette di sapere se un azione associata ad uno blocco IEC è attiva o no; Per questo tipo di step esiste anche una variabile (<**nomedellostep**>**.t**) di tipo TIME che "conta" il tempo in cui il blocco rimane attivo.

SFC Flags: Per il controllo dell'esecuzione di un SFC POU possono essere usati i cosidetti flag, variabili che vengono generate automaticamente da CoDeSys durante l'esecuzione del progetto. Per leggere questi flag bisogna definire appropriate variabili globali o locali come ingressi o come uscite. Ecco alcuni degli SFC flag più utili (per la lista completa consultare il manuale):

SFCInit Quando viene settato al valore `TRUE` l'esecuzione dell'SFC torna allo step iniziale (`Init`), tutti gli altri flags vengono resettati. Il blocco `Init` verrà eseguito solo quando questo flag verrà reimpostato al valore `FALSE`.

SFCPause L'esecuzione dell'SFC rimane bloccata fintanto che questo flag avrà valore `TRUE`.

SFCTrans Questo flag diveta `TRUE` ogni volta che viene attivata una transizione nell'esecuzione dell'SFC.

SFCCurrentStep E' una `STRING` con valore pari al nome dello step attivo in quel determinato istante.

Questi risultano essere molto utili per la gestione pseudo padre/figlio di cui si parlerà nella sezione 3.7.

3.3 Il linguaggio CFC

Il linguaggio CFC (Continuous Function Chart) è un linguaggio introdotto da CoDeSys ma non è presente nella norma IEC 61131-3. Questo linguaggio può essere definito anche come *FreeFBD*, infatti è analogo al linguaggio FBD della norma ma permette di inserire liberamente i vari blocchi in qualsiasi punto dell'editor. Questa proprietà permette di creare diagrammi più complessi e di inserire anche delle retroazioni in maniera diretta.
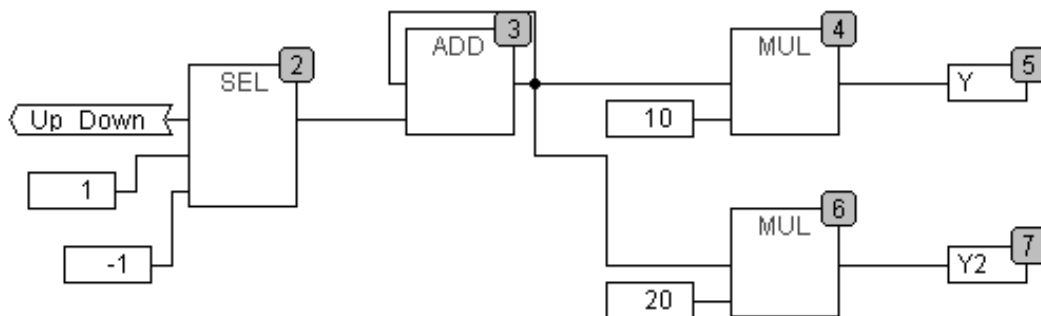
Figure 12: *Esempio di diagramma CFC*

Nel diagramma si possono inserire tutti i tipi di blocchi già esistenti nel linguaggio FBD e anche

tutti i blocchi che sono creati dall'utente insieme a input, output, return, commenti, didascalie e salti. Le linee di connessione tra gli input e gli output dei vari blocchi sono disegnate automaticemnte trascinando il mouse. Verrà sempre disegnata la connessione più corta possibile tenendo conto delle connessioni già disegnate, dei blocchi già inseriti e degli spazi liberi presenti. Inoltre le linee vengo regolate automaticamente se vengono spostati i blocchi di origine o destinazione della connessione stessa.
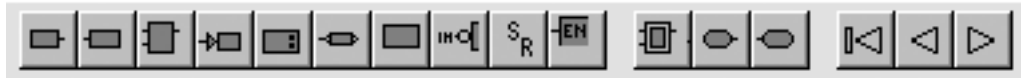
Figure 13: *Pulsanti dell'Editor CFC*

In figura 13 sono indicati i pulsanti dell'Editor CFC dell'ambiente CoDeSys. Nel gruppo di sinistra ci sono tutti quei pulsanti che servono per l'inserimento di tutti i possibili blocchi:

1. Input;
2. Output;
3. Box;
4. Jump;
5. Label;
6. Return;
7. Comment;
8. Negation;
9. Set/Reset;
10. Enable/Enable Output.

Nella parte centrale ci sono tre pulsanti che servono per la definizione di macroblocchi, mentre con i pulsanti di destra ci si può muovere tra i vari livelli di questi macroblocchi per editarne l'interno.

3.4 Tipi di dato

CoDeSys supporta dei tipi di dati standard, a questi tipi di dato sono associati lo spazio in memoria necessaria per allocarli ed il valore che può essere memorizzato.

3.4.1 BOOL

Questi tipi di dato sono le variabili booleane, ad esse possono essere associati i valori TRUE e FALSE. Vengono riservati 8 bit di memoria per queste variabili.

3.4.2 Interi

I tipi di dato interi presenti in CoDeSys sono i seguenti: BYTE, WORD, DWORDS, SINT, USINT, INT, UNIT, DINT e UDINT. Ciascun di questi tipi presenta differenti range per il valore che è possibile memorizzare:

Tipo	Valore minimo	Valore massimo	Spazio allocato in memoria
BYTE	0	255	8 Bit
WORD	0	65535	16 Bit
DWORD	0	4294967295	32 Bit
SINT	-128	127	8 Bit
USINT	0	255	8 Bit
INT	-32768	32767	16 Bit
UINT	0	65535	16 Bit
DINT	-2147483648	2147483647	32 Bit
UDINT	0	4294967295	32 Bit

3.4.3 REAL / LREAL

I tipi di dato REAL e LREAL sono tipi di dato reali a virgola mobile, vengono usati per rappresentare numeri razionali. Per il tipo REAL vengono riservati 32 bit di memoria, il limite inferiore è 1.175494351e-38 ed il limite superiore è 3.402823466e+38 mentre per LREAL vengono riservati 64 bit di memoria, il limite inferiore è 2.2250738585072014e-308 ed il limite superiore è 1.79769313486231e+308.

3.4.4 STRING

Il tipo STRING è il tipo di dato che può contenere qualsiasi tipo di carattere, la dimensione della stringa è definita nella dichiarazione, nel caso non venga definita la dimensione è di 80 caratteri. La lunghezza di una stringa in CoDeSys è illimitata, però le funzioni che lavorano con le stringhe possono elaborare stringhe di al massimo 255 caratteri. Esempio di una dichiarazione di stringa di 35 caratteri:

```
str:STRING(35):='Questa è una stringa'
```

3.4.5 Tipo di dato data

I tipi di dato TIME, TIME_OF_DAY (abbrevazione TOD), DATE and DATE_AND_TIME (abbrevazione DT) sono internamente memorizzate come DWORD, il tempo viene fornito in millisecondi in TIME e TOD, mentre in DATE and DT viene fornito in secondi. La dichiarazione nelle variabili temporali viene eseguita dall'iniziale "t" o "T" seguita da # ed il valore numerico, che può essere definito in giorni (identificato da "d"), ore (identificato da "h"), minuti (identificato da "m"), secondi (identificato da "s") e millisecondi (identificato da "ms"). L'ordine dei valori inseriti deve essere posto come quello indicato, non è obbligatorio inserire tutti i campi, ecco alcuni esempi:
Assegnamenti corretti:

```
TIME1:= T#14ms;
```

```
TIME1:= T#100S12ms; (*Il limite superiore può superare il limite*)
TIME1:=t#12h34m15s
```
Assegnamenti non corretti:
```
TIME1:=t#5m68s; (*Il componente inferiore ha superato il suo limite.*)
t#4ms13d; (*Ordine non corretto*)
```
Il valore massimo che si può inserire è 49d17h2m47s295ms che corrisponde a 4294967295ms

3.4.6 Indirizzi

E' possibile visualizzare o scrivere una singola locazione di memoria, attraverso l'uso di una sequenza di caratteri speciali che è una concatenazione di % con un prefisso per il campo, un prefisso per la dimensione e dei numeri. Il prefisso per il campo viene indicato con: I, per indicare ingressi, Q, per indicare uscite, M, per indicare locazioni di memorie; mentre il prefisso per la dimensione sono: X o niente per indicare singoli bit, B per indicare byte, W per indicare Word e D per indicare una duoble word. Alcuni esempi:
```
%QX7.5 %Q7.5 (*bit uscita 7.5*)
%IW215 (*Word ingresso 215*)
%MD48 (*Double word alla locazione di memoria 48*)
```

3.4.7 Variabili

Nella finestra **Resource** è possibile impostare le variabili globali, attraverso la cartella **Global Variables**. Le variabili globali sono le variabili che vengono riconosciute in tutto il progetto, cioè in tutti i programmi che compongono il progetto; nel caso in cui siano presenti molte variabili è possibile suddividirle in gruppi, cliccando col tasto destro su Global Variables e scegliendo **Add object** si possono inserire delle cartelle per suddividere le variabili globali.
In un progetto è possibile definire una variabile locale che ha lo stesso nome di una variabile globale, in questo caso dentro un POU verrà utilizzata la variabile locale.

3.5 Function block

Il **Function Block** è un tipo di POU di CoDeSys molto simile ad una funzione. Questo ha un certo numero di variabili di ingresso e di uscita ma, a differenza delle funzioni standard, non ha un valore di ritorno. Un Function Block è molto simile ad una classe nella programmazione ad oggetti e, in quanto tale, se ne possono creare molte istanze. Ogni istanza ha un suo identificatore e una sua struttura di variabili di ingresso, di uscita e interne. Le varie istanze possono essere dichiare come variabili globali o locali mentre il nome del Function Block è il tipo di variabile.
Per creare un Function Block non bisogna fare altro che creare una nuova POU e selezionare il pulsante `Function Block` nel dialog, come mostrato in figura 14.
Le variabili di input e di output di una istanza posso essere usate da un'altra POU usando la seguente sintassi:

$$< \text{nome istanza} > . < \text{nome variabile} >$$

Per chiamare una istanza di un function block si possono utilizzare tutti i linguaggi di programmazione dei PLC. I più usati sono comunque lo ST, in quanto è un liguaggio di alto livello, ed il
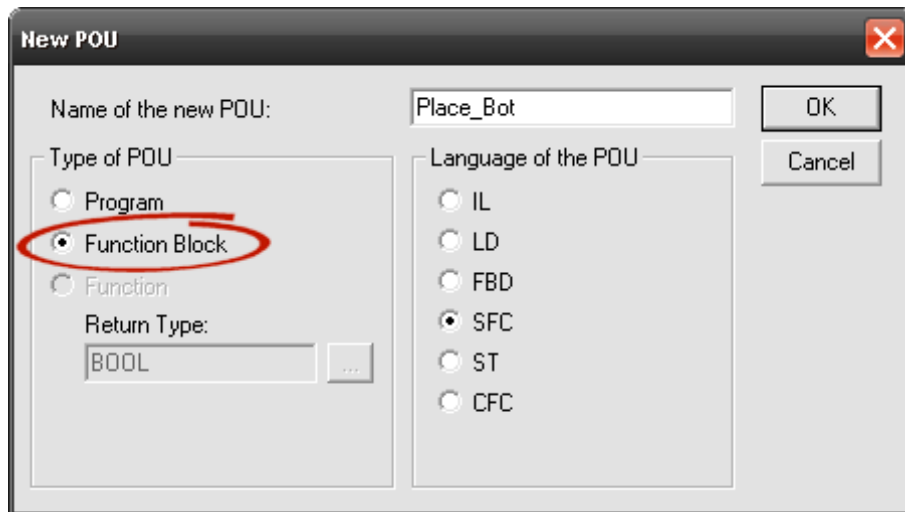
Figure 14: Creazione di un Function Block

(a) Chiamata function block ST.

(b) Chiamata function block LD.
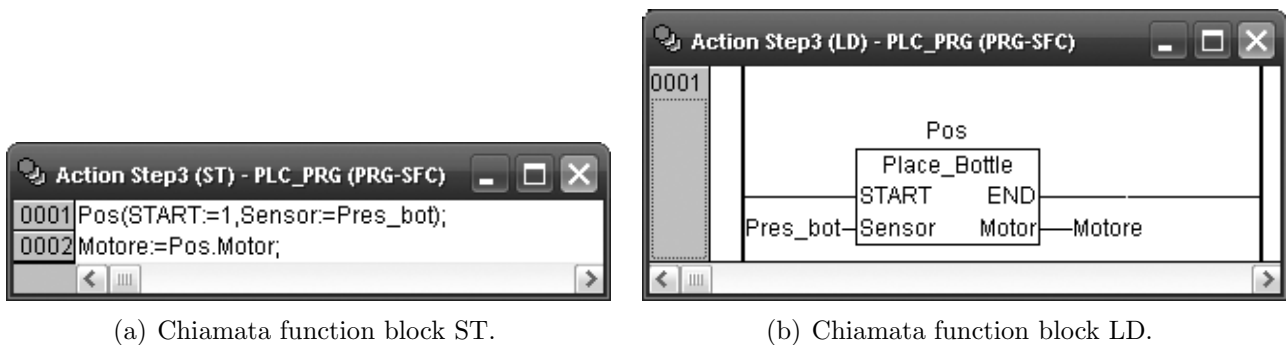
Figure 15: Esempio di chimata di un function block.

linguaggio LD perchè risulta più intuitivo dal punto di vista grafico. In linguaggio ST bisogna scrivere una istruzione del tipo:

$$< \text{nome istanza} > (InputVar1 := val1, InputVar2 := val2, ...);.$$

Se poi si vogliono associare le varibili di uscita ad altre varibili globali o locali al programma chiamante bisogna aggiungere:

$$var1 :=< \text{nome istanza} > .OutputVar1;$$

Per chiamare un function block in linguaggio LD bisogna selezionare `function block` nella finestra che appare cliccando col tasto destro del mouse sulla linea orizzontale e scegliere quello che vogliamo inserire. Una volta inserito il blocco basta indicare i nomi delle variabili da associare agli ingressi e alle uscite a fianco delle variabili di input/output indicate all'interno del rettangolo. Possiamo anche usare gli interruttori e le bobine del linguaggio. Quando si definisce un function block, è come se si definisce una nuova classe di variabili, si definisce un nuovo tipo di variabile. La sua principale caratteristica è che il function block è sempre vivo, cioè mentre una funzione

viene chiamata, ed una volta chiamata non esiste più, il function block esiste per tutta la durata del programma. Per utilizzare i function block in Codesys, una volta averlo definito bisogna definire un programma di tipo `FBD`, cioè creare una nuova POU ti tipo Function Block Diagram. Una volta create questa POU è possibile richiamare i function block creati in precedenza nel seguente modo: una volta generato il programma FBD, appaiono dei punti interrogativi seguiti da un box tratteggiato, cliccando col pulsante destro e selezionando il comando `Box` è possibile inserire un nuovo function block. Una volta cliccato sul comando `Box` appare un blocco, se si evidenzia il nome e si preme il comando F2 (comando per il completamento istruzioni in Codesys) appare una finestra come in figura 16. Selezionando User defined Function Block appare la lista
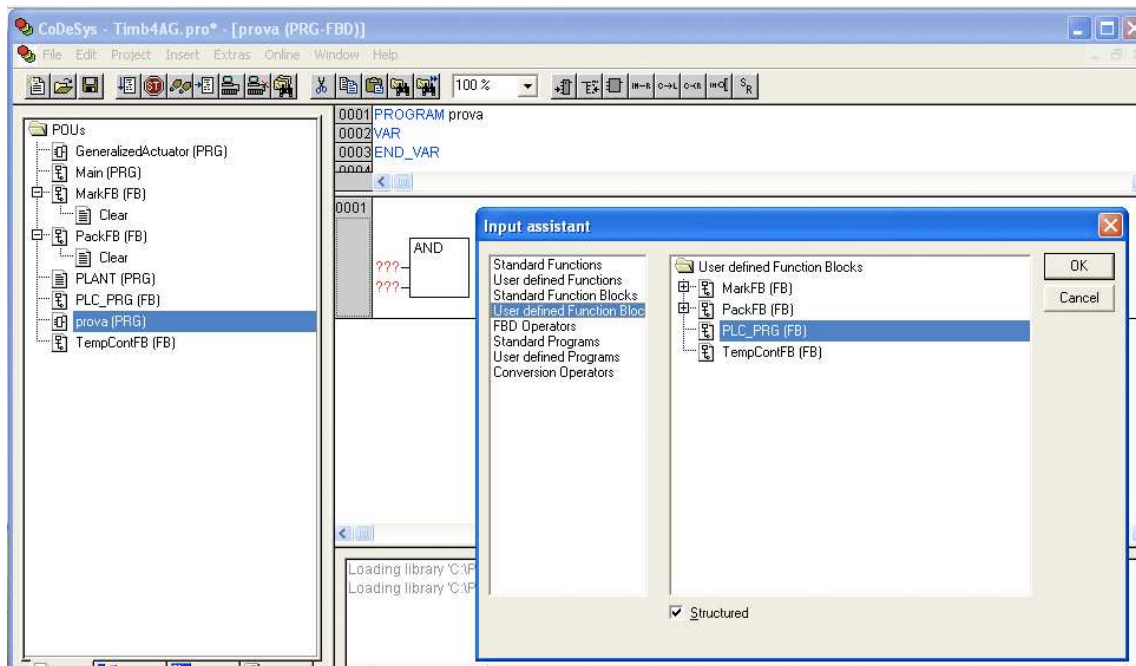
Figure 16: Definizione di un function block

di tutti i function block creati dall'utente, è possibile richiamare quello desiderato, ed associarlo ad una istanza. Alla singola istanza è possibile associare le variabili di ingresso ed uscita. Per definire un nuova istanza, bisogna cliccare con il tasto destro nell'editor ed usare il comando `Network` per inserire una nuova "riga" orizzontale dove poter inserire una nuova istanza di un function block. Alla fine si ottiene uno schema come quello di figura 17, dove sono rappresentati le istanze dei vari function block.
In questo caso i function block sono quelli chiamati PackFB, TempContFB, MarkFB mentre le istanze sono Pack, TempCont e Mark.

3.6 Task Configuration

In precedenza si è visto che quando si crea un progetto da zero, compare il menù per la creazione della prima POU del progetto; di default, la nuova POU è un Program, con il nome PLC_PRG, da programmare in **ST**. Il nome di default non è casuale, infatti se non verranno fatte in seguito modifiche alla configurazione del progetto, una volta lanciata l'esecuzione del progetto
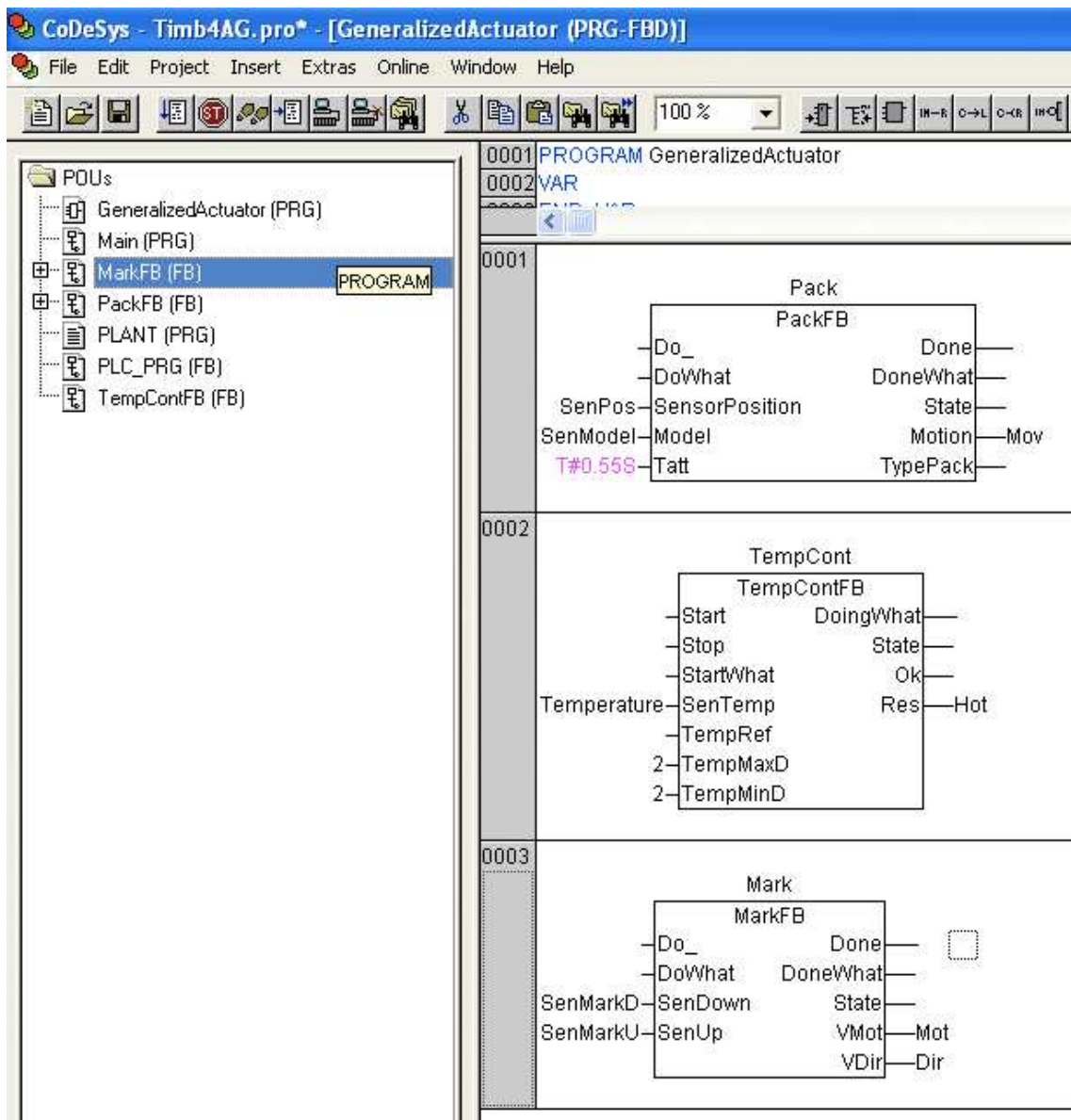
Figure 17: Definizione di istanze di function block
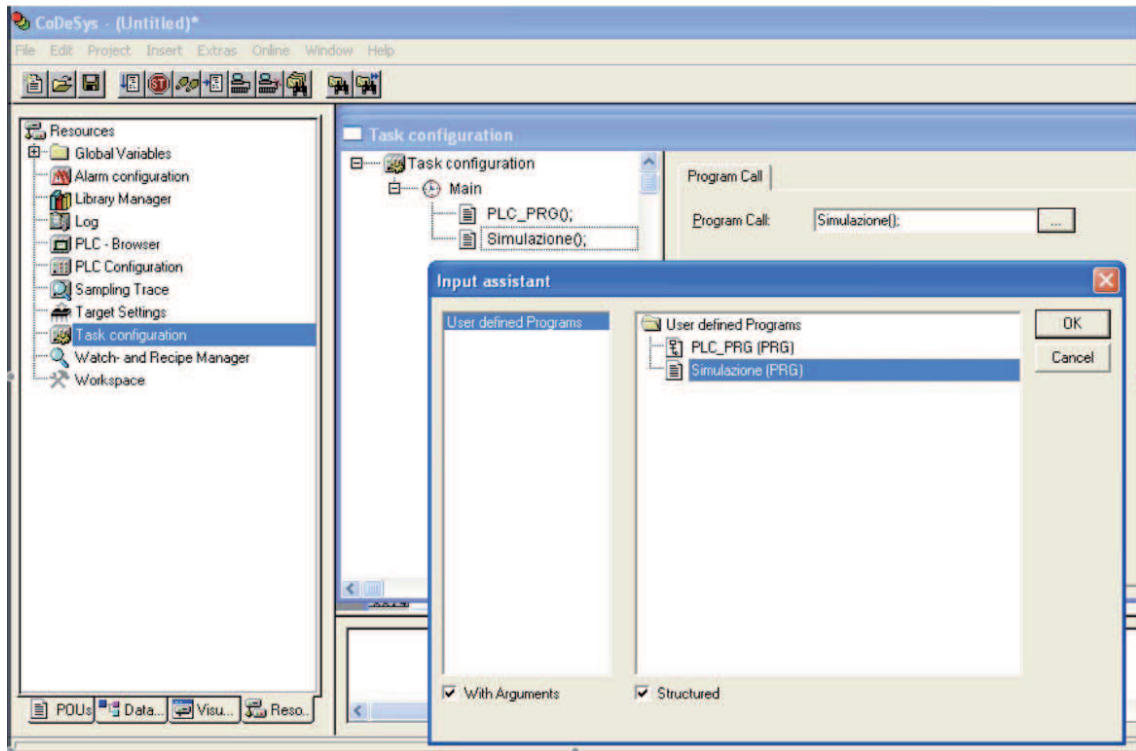
Figure 18: Finestra di configurazione per i Task

l'unica POU eseguita automaticamente dal sistema sarà proprio il Program con il nome chiave PLC_PRG. Tutte le altre POU saranno eseguite solo se richiamate esplicitamente da tale Program.

Nel nostro caso invece vogliamo che sia eseguite più POU, cioè vogliamo che sia eseguita sia la POU dove è presente il programma per il controllo logico, sia la POU dove è presente il programma per la simulazione del Plant, per fare questo dobbiamo modificare il `Task Configuration` situato nel menù `Resources`, come si può vedere dalla figura 18. Supponiamo ad esempio che il programma per il controllo logico sia nella POU chiamata *PLC_PRG* mentre il programma per la simulazione del plant sia nel programma chiamato *Simulazione*. Cliccando due volte su `Task Configuration` appare una nuova finestra da dove è possibile definire nuovi task, cliccando col tasto destro su `Task Configuration` di questa finestra è possibile inserire un nuovo task con il comando `Append Task`. Possiamo ad esempio chiamare questo Task *Main*, a questo punto dobbiamo inserire i programmi che devono essere eseguiti in questo Task, per fare questo si deve cliccare con il tasto destro su *Main* e selezionare il comando `Append Program Call`. In questo modo appare una finestra da dove è possibile inserire il programma, si può digitare il nome oppure cercandolo attraverso l'opzione sfoglia (quella che in figura è rappresentata con dei puntini).

In figura 19 è possibile vedere le opzioni per configuarare i singoli task, è possibile assegnare una priorità di esecuzione, il tipo di esecuzione e la velocità del ciclo di esecuzione. Per tipo di esecuzione si intende quando deve essere eseguito il programma, se ad esempio viene selezionato l'opzione `cyclic`, il programma viene eseguito ciclicamente, è possibile scegliere di fare eseguire il programma tramite eventi. Nelle esercitazioni, e nelle tesine per gli esami, la modalità di
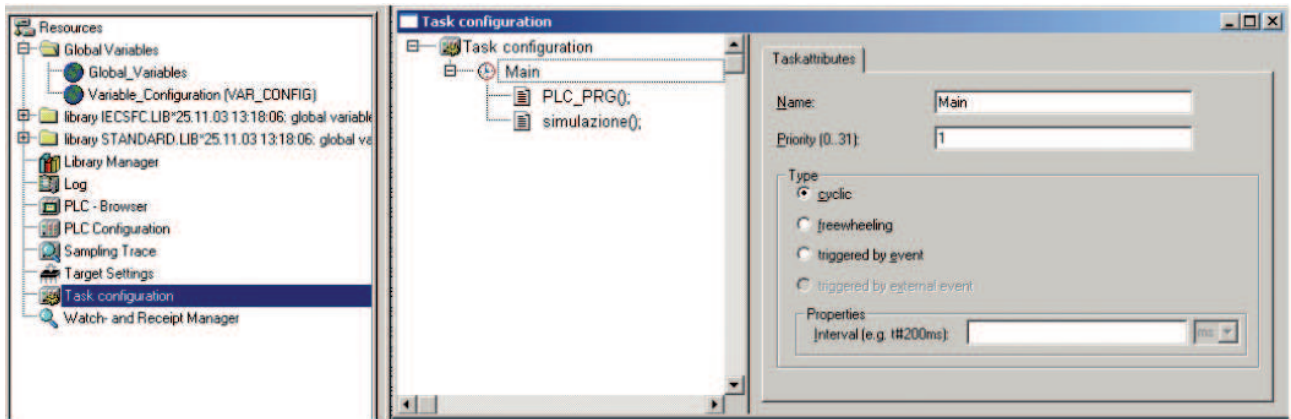
Figure 19: Finestra di configurazione per il singolo Task

esecuzione è `cyclic`.
Per capire come funziona la schedulazione dei programmi in Codesys, andiamo ad implementare
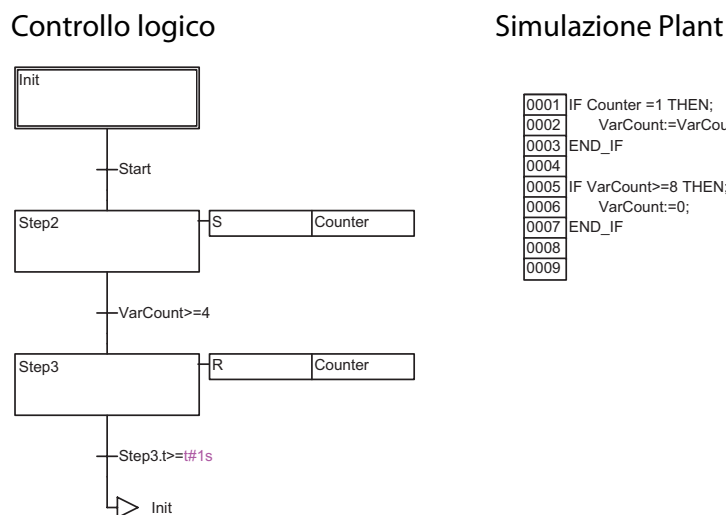
Figure 20: Programmi esempio gestione TASK

un semplice esempio con due programmi, uno che sarà il programma per la logica di controllo ed uno che sarà quello della simulazione del plant. Il programma della logica di controllo, che chiameremo PLC_PRG, quando è premuto un pulsante di start setta ad 1 la variabile counter e quando la variabile VarCount è maggiore di 4 resetta la variabile Counter, per poi aspettare un secondo e ritornare nello stato di Init. Il programma di simulazione del plant, che chiameremo Plant, quando la variabile Counter è a 1 incrementa la variabile VarCount, per poi resettarla quando questa variabile è maggiore uguale ad 8; il codice di questi due programmi è riportato in figura 20. Andiamo a vedere come variano queste variabili, usando una esecuzione passo-passo, a seconda di come impostiamo i parametri nel `Task Configuration`.
Impostiamo un unico task di tipo ciclico dove mettiamo prima PLC_PRG e poi Plant.

Ciclo	Stato	Counter	VarCount
1	Init	FALSE	0
2	Step2	FALSE	0
3	Step2	TRUE	1
4	Step2	TRUE	2
5	Step2	TRUE	3
6	Step2	TRUE	4
7	Step3	TRUE	5
8	Init	FALSE	5
9	Step2	FALSE	5
10	Step3	TRUE	6
11	Init	FALSE	6
12	Step2	FALSE	6
13	Step3	TRUE	7
14	Init	FALSE	7
15	Step2	FALSE	7
16	Step3	TRUE	0
17	Init	FALSE	0

Impostiamo ora sempre un unico task di tipo ciclico dove mettiamo prima Plant e poi PLC_PRG:

Ciclo	Stato	Counter	VarCount
1	Init	FALSE	0
2	Step2	FALSE	0
3	Step2	TRUE	0
4	Step2	TRUE	1
5	Step2	TRUE	2
6	Step2	TRUE	3
7	Step3	TRUE	4
8	Init	FALSE	5
9	Step2	FALSE	5
10	Step3	TRUE	5
11	Init	FALSE	6
12	Step2	FALSE	6
13	Step3	TRUE	6
14	Init	FALSE	7
15	Step2	FALSE	7
16	Step3	TRUE	7
17	Init	FALSE	0

E' possibile vedere come a seconda della sequenza con cui si mettono i due programmi nel ciclo di esecuzione, vari l'andamento delle variabili (si vedano primi tre cicli di esecuzione ad esempio), questo fatto è dovuto che a seconda che venga effettuato prima un programma di un altro, cambia la sequenza di valutazione della variabile e suo incremento. Quando si esegue il programma non cycle to cycle ma in modalità run, sembra che Codesys perda questa diversità di esecuzione e che scheduli i programmi con un suo ordine che non è possibile cambiare, tranne che andando a implementare i programmi su più task e dandogli

tempi di ciclo diversi.

3.7 La gestione Padre/Figlio

La gesione di tipo padre/figlio è una gestione particolare secondo la quale uno o più SFC, denominati figli, sono dipendenti da un altro SFC detto padre. Il padre, e solo lui, ha il potere di far partire o fermare in qualsiasi momento l'esecuzione dei figli. Inoltre i figli, una volta che sono partiti, si evolvono in parallelo al padre.
Nell'ambiente ISAGRAF si possono creare degli SFC figli di altri SFC ed esistono delle funzioni primitive che regolano la gestione di tipo padre/figlio:

GSTART per avviare il programma figlio;

GKILL per terminare l'evoluzione del figlio;

GFREEZE per sospendere l'evoluzione del figlio;

GRST per riavviare l'evoluzione del figlio.

La norma IEC 61131-3 non è molto precisa riguardo a questo punto, impone solo di lasciare la possibilità che ci possano essere delle dipendenze tra diversi diagrammi SFC. L'ambiente CoDeSys è conforme a questa norma riguardo al punto in esame. Infatti non mette a disposizione del programmatore delle funzioni primitive simili alle suddette. Ci sono però alcune variabili che, se gestite in maniera opportuna, permettono di fare anche nell'ambiente CoDeSys una gestione che si può definire *pseudo padre/figlio*. Il metodo più importante e funzionale prevede l'utilizzo dei `flag`, Codesys ha dei nomi predefiniti per questi flag, agendo sui quali è possibile mettere in relazione programmi differenti:

SFCInit Quando viene settato al valore `TRUE` l'esecuzione dell'SFC torna allo step iniziale (`Init`), tutti gli altri flags vengono resettati. Il blocco `Init` verrà eseguito solo quando questo flag verrà reimpostato al valore `FALSE`.

SFCPause L'esecuzione dell'SFC rimane bloccata fintanto che questo flag avrà valore `TRUE`.

SFCTrans Questo flag diveta `TRUE` ogni volta che viene attivata una transizione nell'esecuzione dell'SFC.

SFCCurrentStep E' una `STRING` con valore pari al nome dello step attivo in quel determinato istante.

Se, infatti, questi flag vengono elencati tra le `VAR_INPUT`, cioè le variabili di ingresso, di un programma, questi possono essere modificati da qualsiasi altro programma presente nel progetto. Usando opportunamente i flag `SFCInit` e `SFCPause` si può, quindi, ricreare una gestione che è simile a quella padre/figlio di ISAGRAF.
Questa non è una vera e propria gestione padre/figlio per due motivi. Innanzi tutto, in teoria, solo il padre potrebbe avviare e terminare il figlio, invece i flag di sistema possono essere modificati da qualsiasi programma presente nel progetto; è come se tutti fossero padri dei figli. Si può ovviare a questo problema stando attenti a fare in modo che un solo programma usi queste variabili

sul figlio, anche se tutti gli altri avrebbero la potenzialità di farlo. Il secondo motivo riguarda il fatto che quando il padre esegue l'azione `GKILL`, il figlio scompare proprio mentre, con l'uso di `SFCInit`, il figlio rimane presente anche se congelato nella posizione di `Init` fino a quando il flag suddetto non ritornerà al valore di `FALSE`. Da un punto di vista dell'utente questa non è una grossa differenza, anche perchè non si nota nel funzionamento del sistema, è però un fattore importante dal punto di vista concettuale.
Un altro possibile metodo è quello di inserire un SFC all'interno dello stato di un altro SFC. L'esecuzione è, ovviamente, strettamente dipendente dall'esecuzione del primo, però, in questo caso, non rispetta la proprietà secondo la quale il figlio può evolversi indipendentemente dal padre. Infatti, in questo caso, se il padre cambia stato il figio non viene più eseguito. Questo metodo non è più utilizzabile se lo stato dell'SFC padre in cui il figlio andrebbe terminato è distante uno o più stati da quello in cui il figlio viene avviato. E' invece possibile utilizzarlo se durante l'esecuzione del figlio, il padre deve stare fermo in un unico stato, cosa che però non sempre avviene.
Per capire meglio come funzione la gestione pseudo padre/figlio a flag si è creato un piccolo progetto formato da due diagrammi SFC: `Padre` e `Figlio` la cui struttura è visibile in figura 21.

`Figlio` fa solamente tre azioni denominate `var0`, `var1` e `var2` le quali salvano rispettivamente il valore 0, 1 e 2 in una variabile globale chiamata *number*. Una volta che è avviato rimane nello stato `Init` per 5 secondi poi cicla saltando tra gli stati 2 e 3 ogni 2 secondi. Ho dovuto usare degli IEC-Steps perchè gli Step normali non hanno la variabile *.t* associata che calcola il tempo di attivazione. Il `Padre` invece è pilotato dall'esterno con 4 variabili (*Start*, *Stop*, *Pause* e *Restart*) che ne determinano l'evoluzione e le azioni sopra il figlio.
Quando il padre riceve lo *Start* attiva il figlio rimettendo a `FALSE` la variabile *SFCInit* dello stesso. A questo punto il figlio comincia la sua evoluzione. All'occorrenza di uno *Stop* il padre ritorna nello stato di `Init` e disattiva il figlio rimettendo il valore `TRUE` nella variabile *SFCInit*. All'occorrenza di un *Pause* il padre congela il figlio agendo sulla variabile *SFCPause* dello stesso e si mette ad aspettare un *Restart* che può far ripartire il figlio da dove si era fermato o uno *Stop* che ha gli stessi effetti di prima.
Tutti questi comandi vengono forniti tramite pulsanti nell'oggetto grafico. Nella visualizzazione ho anche plottato un grafico con l'evoluazione temporale della variabile *number*, il cui risultato è indicato in figura 22 insieme ad alcune frecce che indicano quando e quali comandi sono stati impartiti al padre.
Si può notare che, una volta impartito lo *Start*, si hanno i primi secondi con la variabile a 0 poi inizia la corretta evoluzione del figlio che cambia valore alla variabile ogni 2 secondi. All'occorrenza di un *Pause* il figlio si blocca nello stato in cui era quindi il valore della variabile non cambia fino ad un successivo *Restart*. Alla fine della simulazione si incontra però un fatto molto importante. Quando arriva lo *Stop* la variabile non ritorna a 0 ma rimane ad uno degli altri valori. Questo si può spiegare facilmente ricordando gli effetti di *SFCInit* su un diagramma SFC. Infatti quando questo flag viene impostato a `TRUE`, lo stato iniziale diventa quello attivo ma non viene eseguito fino a quando il suddetto flag non ritorna al valore `FALSE`. Il risultato della simulazione combacia perfettamente con quanto detto, infatti quando arriva lo *Stop* lo stato di `Init` diventa attivo ma non viene eseguito, quindi la variabile globale mantiene l'ultimo valore con la quale era stata settata. All'occorrenza del successivo *Start* viene eseguito lo stato `Init` quindi *number* riottiene il valore nullo per 5 secondi per poi riiniziare la sua evoluzione tra i
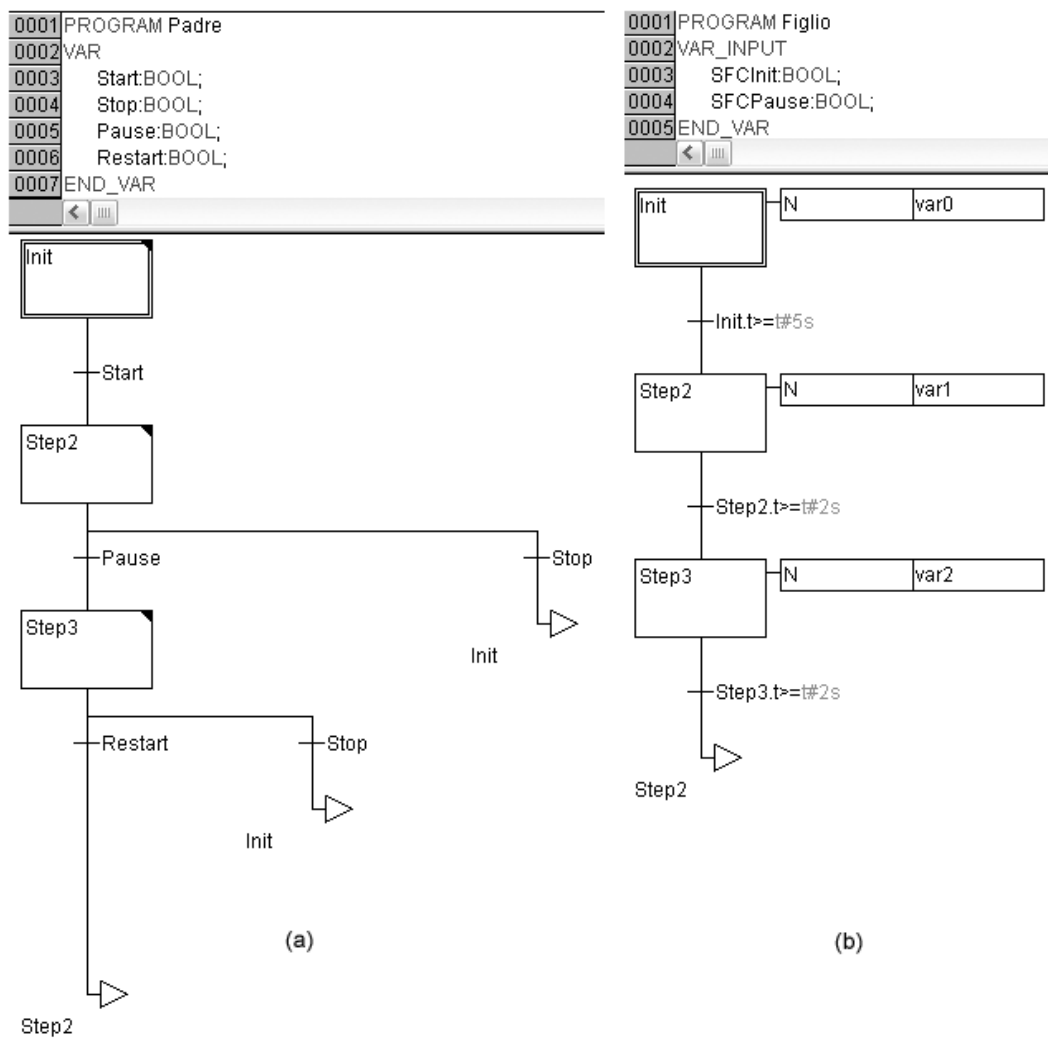
Figure 21: *Diagrammi SFC di* ***`Padre(PRG)`*** *(a) e* ***`Figlio(PRG)`*** *(b)*
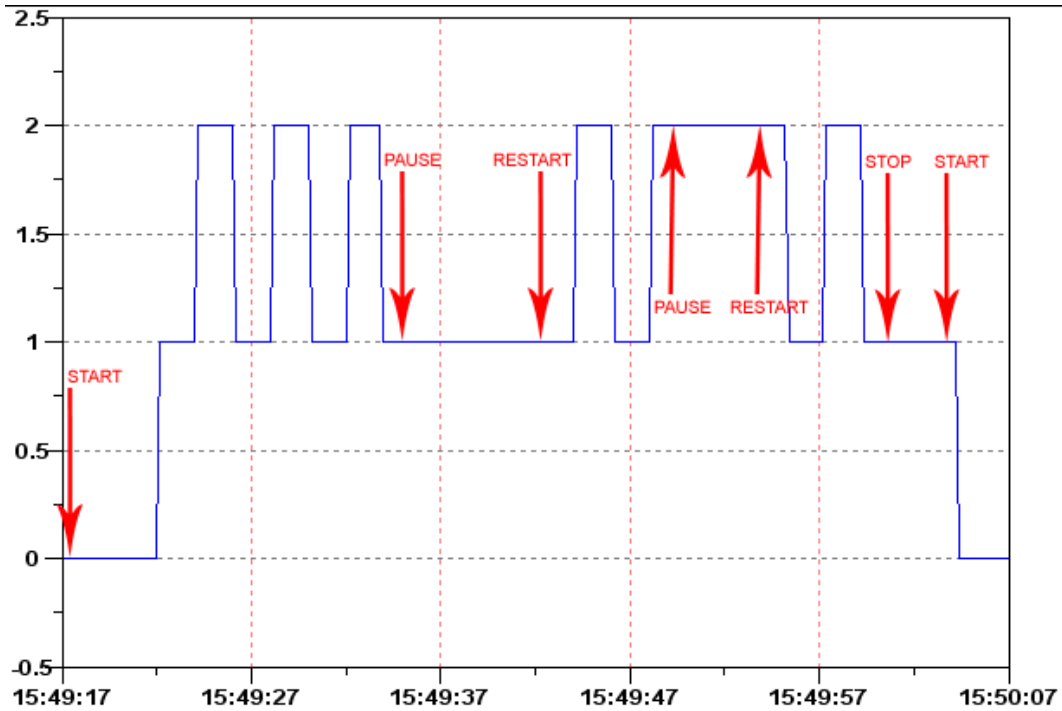
Figure 22: *Risultato della simulazione dei programmi* `Padre(PRG)` *e* `Figlio(PRG)`

valori 1 e 2.
Per fare funzionare correttamente questo progetto bisogna andare nel `Task Configuration` e creare un nuovo task che faccia partire entrambi i programmi: sia `Padre` che `Figlio`. Questo perchè in genere CoDeSys fa partire di default solo la POU `PLC_PRG`, quindi se vogliamo fare partire più POU contemporaneamente o anche una sola con un nome differente bisogna creare un nuovo task. In alternativa si potrebbe anche richiamare il diagramma figlio con una chiamata simile a quelle per i function block all'interno di uno step del padre, ma questo metodo provoca una errata evoluzione del diagramma figlio. Infatti quando si esce dalo step dove viene chiamato il figlio si blocca nell'ultimo step eseguito, qualsiasi sia il valore dei flag del sistema. In questo esempio, l'evoluzione dal punto di vista esterno non ha delle differenze, ma rimane solo un problema concettuale. In casi più complessi però potrebbe portare anche a grossi errori nel funzionamento del progetto.
Questa simulazione ha messo in luce un fattore importante nella gestione pseudo padre/figlio in CoDeSys. Quando disattiviamo un figlio le variabili non vengono resettate perchè lo stato `Init` non viene eseguito fino alla successiva riattivazione. Quindi se vogliamo che alcune variabili del figlio vengano resettate con la sua disattivazione dobbiamo farlo esplicitamente dal diagramma padre. Nel nostro caso, mettendo una espressione del tipo

$$\text{number} := 0;$$

all'interno dello stato `Init` del padre, all'occorrenza dello *Stop* il grafico sarebbe ritornato al valore nullo.

4 L'ambiente Codesys: l'interfaccia grafica

Il tool di sviluppo contiene al suo interno un editor integrato per la visualizzazione che permette all'utente di creare oggetti grafici collegati al programma, in modo da sviluppare una interfaccia grafica per la simulazione dell'applicazione. L'integrazione offerta in CoDeSys permette che l'accesso alle variabili sia diretto, e che non ci sia bisogno di configurare OPC o DDE layer, che spesso sono difficoltosi da configurare, perchè la comunicazione viene effettuata con lo stesso meccanismo usato per la programmazione. La visualizzazione può essere effettuata in quattro modi:

- Direttamente dal programma, quando si opera in simulation mode si può immediatamente visualizzare l'interfaccia grafica collegata al controllore, senza aver bisogno di altri tool.
- Attraverso il programma CoDeSys HMI, che è il sistema run time necessario per la visualizzazione, il sistema permette all'utente di creare un software a basso costo per operare sulla macchina usando la visualizzazione generata in CoDeSys.
- Visualizzazione via web, CoDeSys genera una descrizione nel formato XML che viene caricato nel controllore insieme ad un Java-Applet, e che può essere visualizzato attraverso TCP/IP su un browser.
- Per controllori con display integrato la visualizzazione dei dati può essere caricata insieme all'applicazione sul sistema obiettivo, questa soluzione può essere applicata con poca fatica su qualsiasi dispositivo programmabile in CoDeSys.

Per visualizzazione si intende una rappresentazione grafica delle variabile del progetto, l'editor di visualizzazione all'interno di CoDeSys, che è una parte integrante del tool di sviluppo, fornisce elementi grafici che possono essere modificati e collegati alla variabili di progetto, in questo modo è quindi possibile collegare gli ingressi del PLC alla tastiera o al mouse, e di visualizzare le variabili di uscita; le proprietà del singolo elemento per la visualizzazione vengono definite attraverso appropriate finestre di configurazione.

4.1 Editor grafico

Per aprire l'editor grafico di CoDeSys bisogna aprire il browser di progetto nella parte **Visualizations**, la finestra sarà divisa in due parti, nella parte destra è presente lo spazio per collocare gli oggetti grafici, mentre nella parte sinistra è presente il browser di progetto dove è posta la lista delle POU.

Per creare un nuovo Object, basta cliccare col tasto destro su `Visualization` $\longrightarrow$ `Add object` come è possibile vedere dalla figura 23, dalla stessa figura è possibile vedere nella barra degli strumenti gli oggetti che possono essere inseriti.

Dopo aver selezionato un elemento dalla barra, per visualizzarlo sarà necessario tenere premuto il tasto sinistro del mouse e creare la forma dell'elemento in modo da definirne la dimensione approssimativa.

- Attraverso il comando **'Extras'Elementlist'** viene aperta una finestra che mostra la lista degli elementi contenuti nella visualizzazione corrente. In questo elenco gli elementi sono
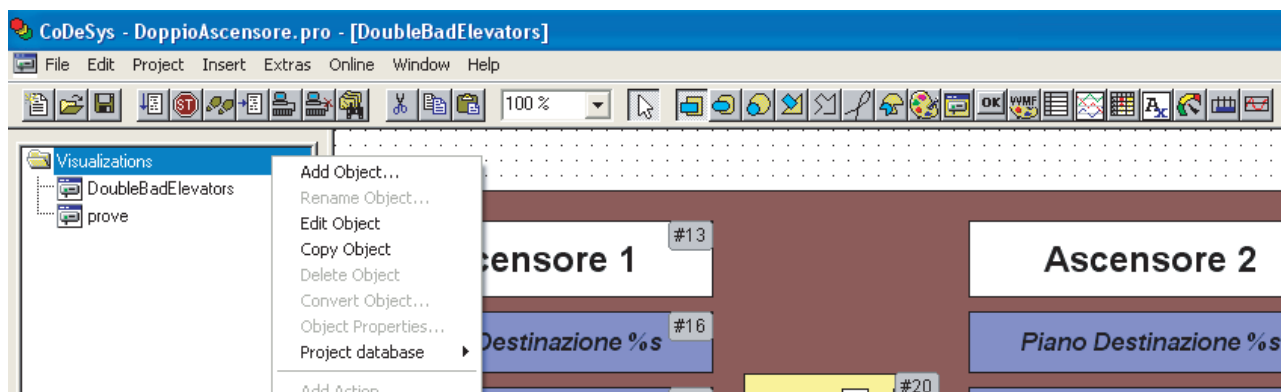
Figure 23: *Editor grafico*

posizionati in ordine di visualizzazione; ovvero se due elementi risultano sovvrapposti nel campo di visualizzazione allora sarà mostrato in primo piano l'elemento contrassegnato da un numero maggiore.
Sono disponibili ovviamente una serie di pulsanti che permettono di modificare la posizione relativa dei vari elementi.
E' possibile inoltre specificare con precisione la posizione di un elemento selezionato dalla lista sfruttando i quattro campi di testo posti nella parte inferiore della finestra stessa.

4.2 Configurazione di un elemento

Attraverso il comando **'Extras'Configure'**, viene aperta la finestra di configurazione di un qualsiasi elemento selezionato ed inserito all'interno della visualizzazione (è possibile accedere a questo dialog anche cliccando due volte sull'elemento stesso). Nella parte sinistra di questa finestra sono presenti le varie categorie disponibili a seconda del tipo di elemento. Scegliendo una particolare categoria vengono mostrati nella parte destra gli eventuali campi da riempire con opportune variabili definite in precedenza oppure le varie opzioni da settare per definire le proprietà dell'elemento in modalità on-line.
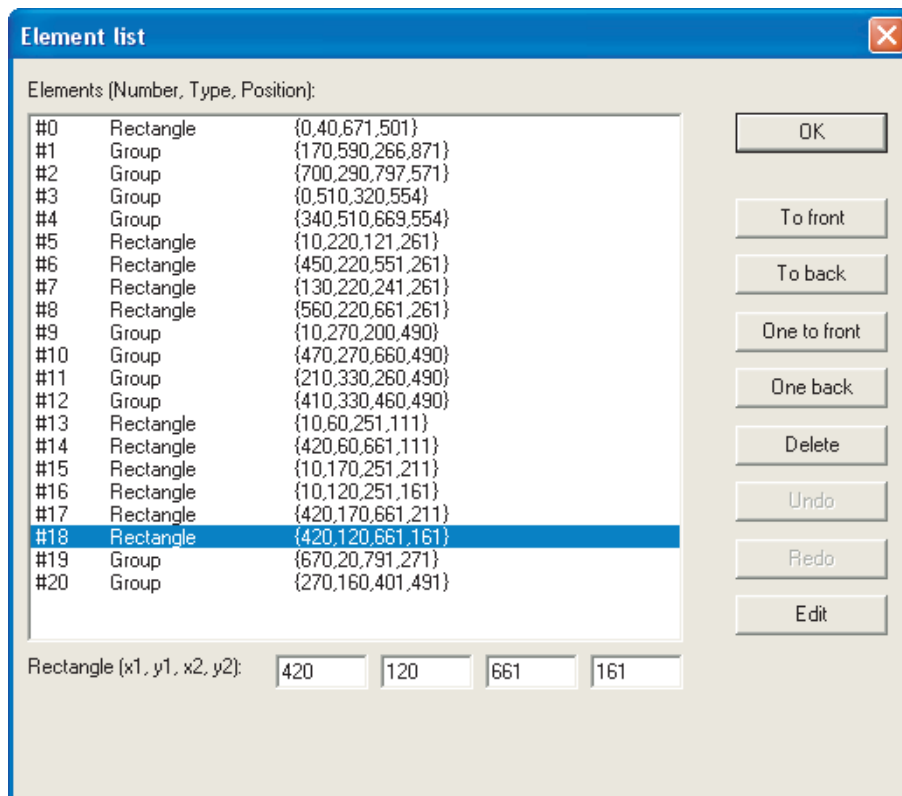
Figure 24: *Finestra* ***'Extras"Elementlist'***
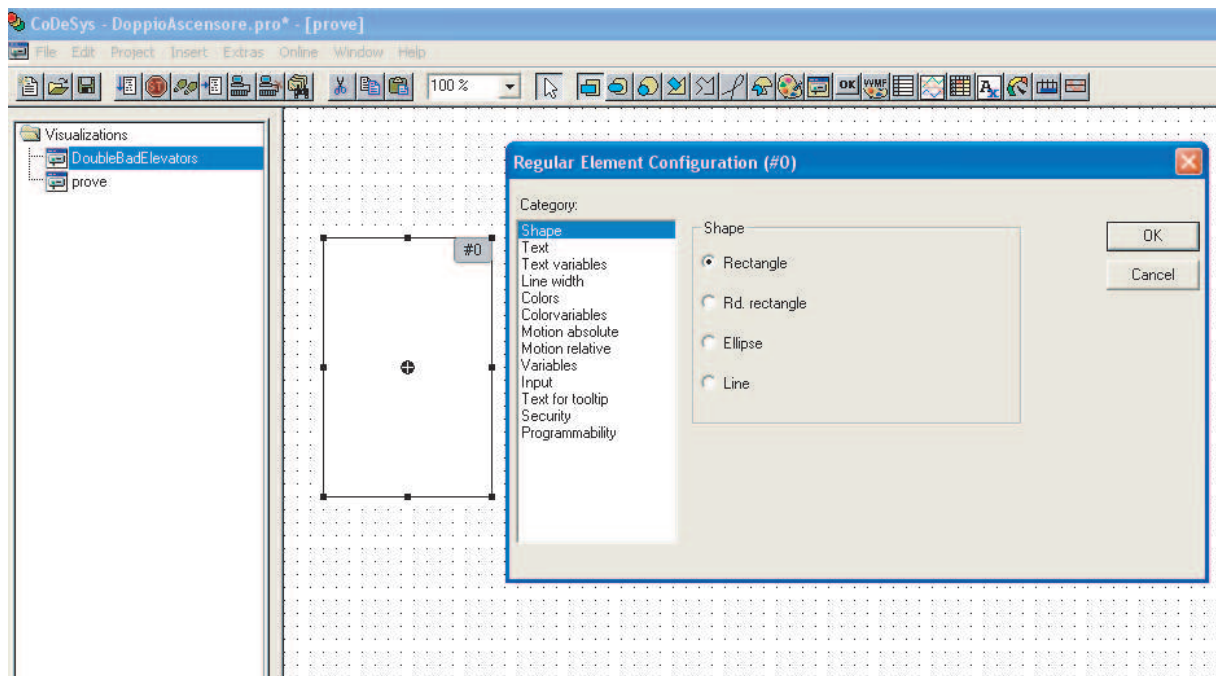
Figure 25: *Finestra di configurazione dell'elemento di visualizzazione*

Si procede ora all'analisi delle singole categorie a disposizione in modo da descrivere le proprietà che possiamo dare ad un particolare elemento.

4.2.1 Shape

Selezionando **'Shape'** è possibile configurare la forma dell'elemento (vedi figura 25).

4.2.2 Angle

Questa categoria è disponibile solo nella configurazione di un elemento circolare (**'Configure Pie'**). E' possibile infatti definire il valore iniziale (**'Start Angle'**) e quello finale (**'End angle'**) dell'angolo descritto dall'elemento.

4.2.3 Text

In questa categoria è possible specificare un testo per l'elemento. Questo può essere inserito direttamente digitando nel campo denominato **'Content'**.

- Se si vuole visualizzare il valore di una particolare variabile definita precedentemente è necessario inserire nel sudetto campo "%" più un carattere a seconda del tipo di variabile che si vuole inserire (vedi la tabella sottostante). Contestualmente il nome della variabile deve essere inserito nel campo denominato **'Textdisplay'** della categoria **'Variables'**.

CARATTERE	**TIPO DI DATO**
d , i	numero decimale
o	numero ottale senza segno
x	numero esadecimale senza segno
u	numero decimale senza segno
c	singolo carattere
s	stringa
f	numero reale

- E' possibile inserire nel campo di testo un "%t" seguito da una certa sequenza di speciali puntatori (placeholders) che vengono rimpiazzati in modalità on-line dalla temporizzazione del sistema. A pagina 13 e 14 del manuale "The CoDeSys Visualization" [?] è presente l'elenco di questi speciali puntatori e viene mostrato l' effetto grafico prodotto da ognuno di essi. La disposizione dei placeholders definisce il formato in cui la temporizzazione deve essere visualizzata.
- Se si include "%¡PREFIX¿" nel testo, è possibile inserire invece di "PREFIX" una certa stringa, che fungerà da identificatore nell'uso di un testo dinamico. Il prefisso sarà usato insieme ad un numero ID, che deve essere definito nel campo 'Textdisplay' della categoria 'Variables'. Questa coppia di valori si riferisce ad un certo testo, che è contenuto in un xml-file fra i possibili testi dinamici. Così in modalità on line verrà visualizzato il testo indicato dalla corrente combinazione ID-prefisso. (Per maggiori informazioni consultare il manuale).
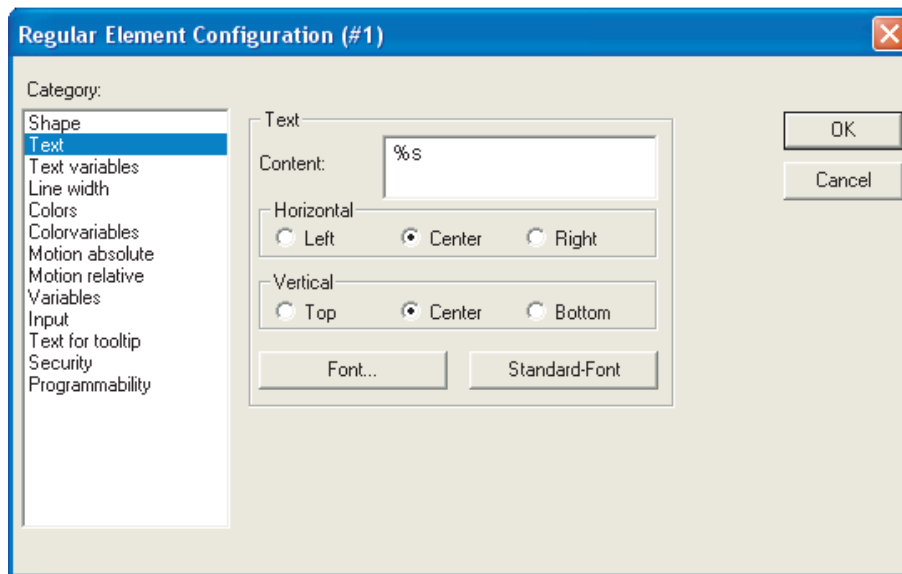
Figure 26: *Finestra di configurazione dell'elemento visualizzato (Categoria Text)*

4.2.4 Textvariables

Nella categoria **'Textvariables'** è possibile specificare una variabile che setti dinamicamente il colore e il formato della stringa definita nella categoria **'Text'**. Inoltre è possibile usare componenti della struttura *VisualObjectType* per definire le proprietà del testo. Per informarsi sui valori e sugli effetti dei componenti di queste strutture vedi la categoria **'Programmability'**. Sotto viene mostrata una tabella di possibili dichiarazioni di variabili che possono essere usate in questa categoria e la rispettiva finestra di configurazione che utilizza queste variabili.

Parameter:	**Meaning:**	**Example entry of project variable:**	**Example Usage of variable in program:**	**corresponding component of structure VisualObjectType:**
Textcolor:	Text color	"plc_prg.var_textcolor"	var_textcolor=16#FF00FF → Farbe	dwTextColor
Textflags:	Alignment (right, left, centered...)	"plc_prg.textpos"	textpos:=2 → Text right justified	dwTextFlags
Fontheight:	Font height in Pixel	".fonth"	fonth:=16; → Font height 16 pt	ntFontHeight
Fontname:	Font name	"vis1.fontn"	fontn:=arial; → Arial is used	stFontName
Fontflags:	Font display (bold, underlined, italic...)	"plc_prg.fontchar"	fontchar:=2 → Text will be displayed bold	dwFontFlags

4.2.5 Line width

In questo box di configurazione è possibile scegliere la larghezza delle linee che delimitano l'elemento. In alternativa si può inserire nel campo **'Variable for line width'** una variabile che
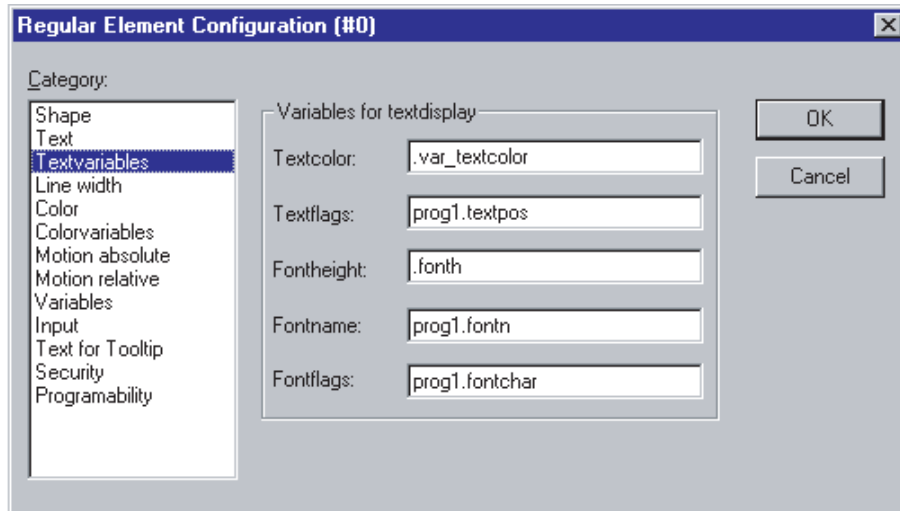
Figure 27: *Finestra di configurazione dell'elemento visualizzato (Categoria Textvariables)*

setti questo parametro dinamicamente.
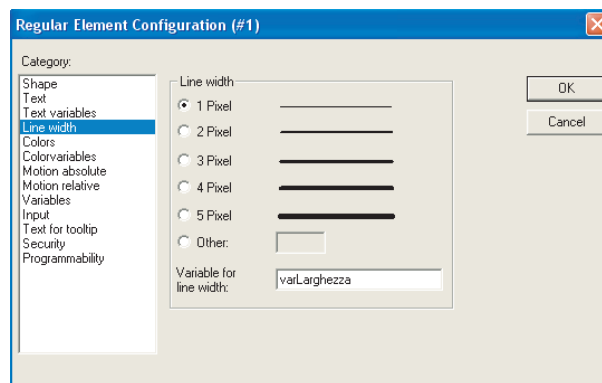
Figure 28: *Finestra di configurazione dell'elemento visualizzato (Categoria Line width)*

4.2.6 Color

In questa categoria si può specificare il colore che l'elemento assumerà in modalità on line. Infatti una volta inserita una variabile booleana nel campo **'Change Color'** della categoria **'Variables'**, l'elemento assumerà il colore settato nel box denominato **Color** quando la variabile sarà FALSE. Viceversa quando il boolean è TRUE, verrà visualizzato con il colore scelto nel box **Alarm Color**.

4.2.7 Color Variables

In questa categoria è possibile inserire eventuali variabili (definite precedentemente all'interno del progetto) attraverso il quale venga modificato dinamicamente il colore interno e/o dei bordi dell'elemento. In alternativa l'aspetto dell'elemento può essere programmato con l'aiuto dei
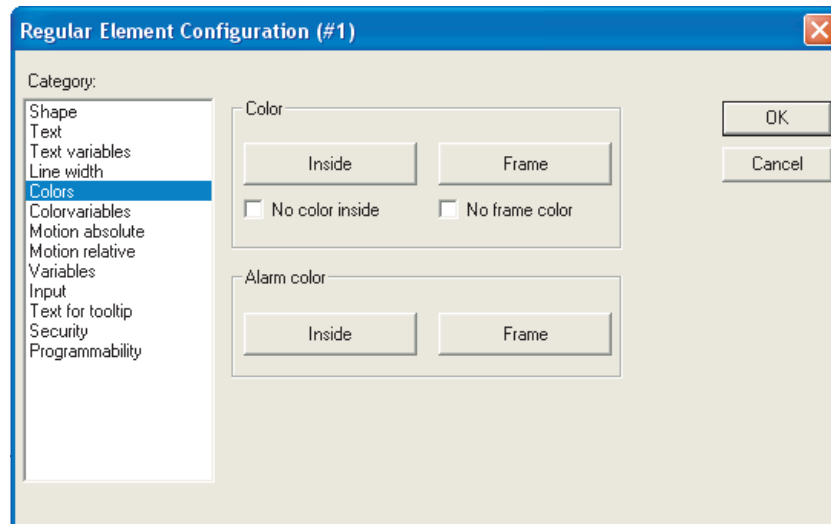
Figure 29: *Finestra di configurazione dell'elemento visualizzato (Categoria Color)*

componenti della struttura *VisualObjectType* (vedi categoria **'Programmability'**). La tabella sottostante mostra come si effettua la dichiarazione di variabili che settano il colore dell'elemento nonchè la lista dei componenti del *VisualObjectType* corrispondenti.

Parameter:	Description:	Example of an entry:	Example for using the variable in the program:	corresponding component of structure VisualObjectType:
Fillcolor:	fill color	"plc_prg.var_fillcol"	var_var_fillcol:=16#FF00FF → fill color pink	dwFillColor
Fillcolor alarm:	fill color if the 'Change color' variable is TRUE	"plc_prg.var_fillcol_a"	var_fillcol_a:=16#FF00FF → alarm fill color pink	dwFillColorAlarm
Framecolor:	frame color	"plc_prg.var_framecol"	var_framecol::=16#FF00FF → frame color pink	dwFrameColor
Framecolor alarm:	frame color if the 'Change color' variable is TRUE	"plc_prg.var_framecol"	var_framecol:=16#FF00FF → alarm frame color farbe pink	dwFrameColorAlarm
Fillflags:	The current inside color configuration can be activated (FALSE) resp. deactivated (TRUE)	"plc_prg.var_col_off"	var_col_off:=1 → the color definition for the fill color will not be regarded, that for the frame remains valid	dwFillFlags
Frameflags:	Display of the frame (solid, dotted etc.)	"plc_prg.var_linetype"	var_linetype:=2; → frame will be displayed as dotted line	dwFrameFlags
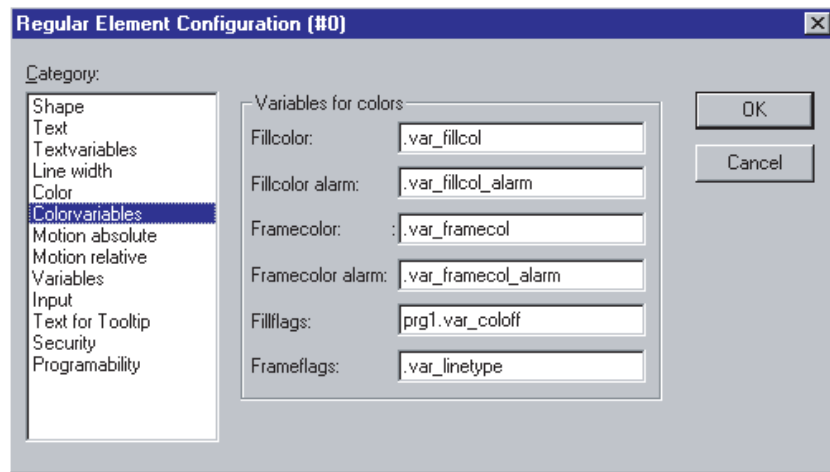
Figure 30: *Finestra di configurazione dell'elemento visualizzato (Categoria Color Variables)*

4.2.8 Motion absolute

In questo dialog di configurazione è possibile inserire delle variabili il cui valore regolerà nella modalità on line uno spostamento dell' intero elemento, lungo l'asse delle ascisse (campo **X-Offset**) e/o lungo l'asse delle ordinate (campo **Y-Offset**). Il valore di un eventuale variabile
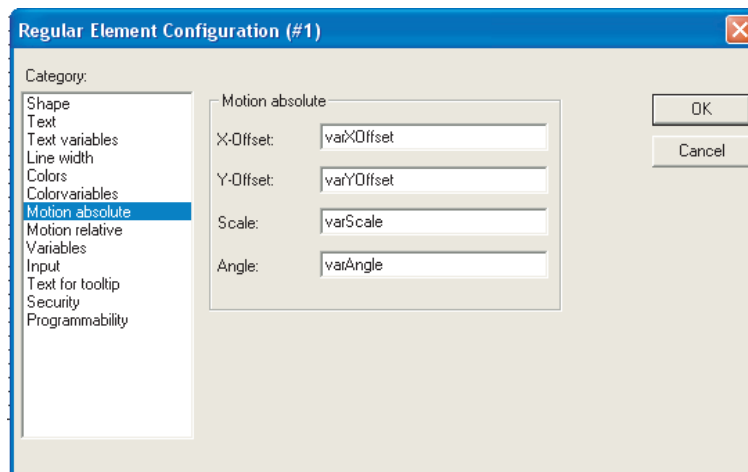
Figure 31: *Finestra di configurazione dell'elemento visualizzato (Categoria Motion Absolute)*

inserita nel campo **Scale** determinerà invece dinamicamente la dimensione dell'elemento. Questo valore funziona quindi da fattore di scala che sarà diviso automaticamente per 1000, in modo che non sia necessario usare delle variabili di tipo REAL per generare una riduzione di scala dell'elemento.
Una variabile nel campo **Angle** causerà il ruotare dell'elemento intorno al cosidetto 'turning point', dipendente ovviamente dal valore della variabile stessa. Il valore è valutato in gradi. Il 'turning point' appare dopo un singolo click sull'elemento, ed è visualizzato con un piccolo cerchio nero con una croce nel mezzo. E' possibile spostare il turning point trascinandolo con il mouse tenendone premuto il tasto sinistro. Inoltre a questo punto fa riferimento anche il cambio

di dimensione dell'elemento nel caso in cui sia presente anche una varibiale all'interno del campo **Angle**.

4.2.9 Motion relative

Nella categoria **'Motion relative'** sono disponibili quattro campi di inserimento relativi ai quattro margini che formano l'elemento (**Left edge**, **Top edge**, **Right edge**, **bottom edge**). I valori delle eventuali variabili (di tipo INT) inserite nei campi permettono di generare un movimento relativo e indipendente di ogni singolo lato dell'elemento e quindi anche di più lati contemporaneamente. La posizione di base di ogni margine dell'oggetto rappresenta lo zero a cui riferire il valore della variabile che eventualmente ne regolerà il movimento in modalità on line. E' necessario inoltre precisare che il valore della variabile è quantizzato nel campo di visualizzazione in pixel.
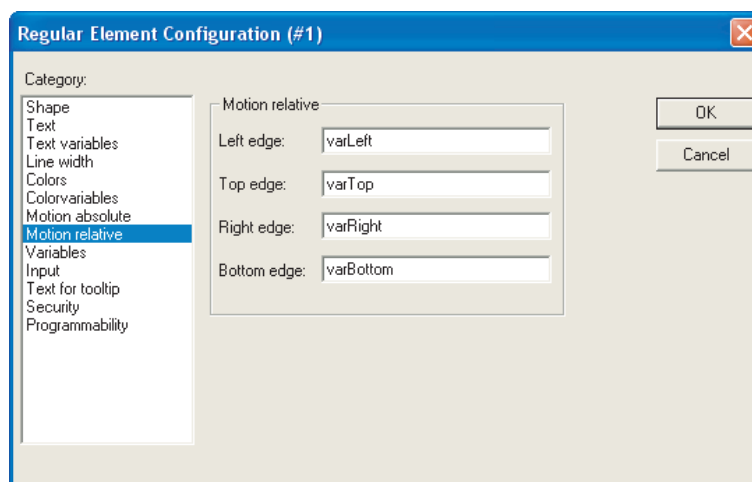
Figure 32: *Finestra di configurazione dell'elemento visualizzato (Categoria Motion relative)*

4.3 Variables

In questa categoria vengono dichiarate tutte le variabili che caratterizzeranno l'aspetto e le proprietà dell'elemento in modalità on line.
Sono disponibili cinque campi:

Invisible: Quando la variabile booleana qui inserita è FALSE, l'elemento sarà visibile, viceversa se è TRUE sarà invisibile.

Disable Input: Se la variabile inserita è TRUE, tutte le impostazioni della categoria **'Input'** verranno ignorate.

Change Color: Se la variabile booleana presente all'interno di questo campo ha il valore FALSE, l'elemento assumerà il colore di default. Se la variabile è TRUE, l'elemento assumerà il colore scelto nel box **Alarm Color** della categoria **'Color'**.
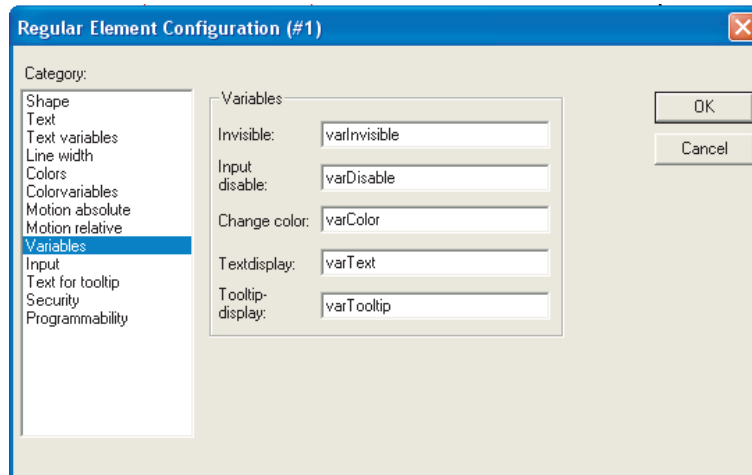
Figure 33: *Finestra di configurazione dell'elemento visualizzato (Categoria Variables)*

Textdisplay:
- Se è stato inserito un "%s" nel campo **Content** della categoria **'Text'**, allora il valore della variabile che è stata definita in **Textdisplay** sarà visualizzato in modalità on line al posto del "%s".
- Se è stato inserito un "%PREFIX" (dove "PREFIX" è un prefisso,una sequenza di lettere) nel campo **Content** della categoria **Text**, allora la variabile (o il valore numerico) inserito qui in 'Textdisplay' sarà interpretato come un ID, che in combinazione con il prefisso crea un riferimento ad un testo contenuto in un XML-file. Questo testo verrà visualizzato al posto di "%PREFIX".
- E'possibile digitare con la tastiera in modalità on line il valore da attribuire alla variabile inserita in questo campo utilizzando **Text Input of variable 'Textdisplay'** della categoria **'Input'**.

Tooltip-display: In questo campo è possibile inserire una variabile di tipo STRING il cui valore vuole essere visualizzato in un tooltip dell'elemento visibile in modalità on line (vedi la categoria **'Text for Tooltip'**).

4.3.1 Input

Nella categoria **Input** devono essere inserite tutte le variabili il cui valore vuole essere modificato in online mode agendo sull'elemento.

Toggle Variable: Se questa opzione viene attivata è possibile inserire nell'apposito campo il nome di una variabile booleana il cui valore può essere modificato in modalità on line. Più precisamente cliccando una volta sull'elemento il valore passa da TRUE a FALSE e ritorna ad essere TRUE inseguito ad un successivo clicco.

Tap Variable: Anche questa opzione permette di modificare il valore della variabile booleana data nel campo d'ingresso ma funziona in maniera diversa. Infatti in questo caso il valore della variabile cambierà solo istantaneamente poichè il valore iniziale verrà ripristinato non appena il pulsante del mouse viene rilasciato. Se la variabile non è stata inizializzata (in
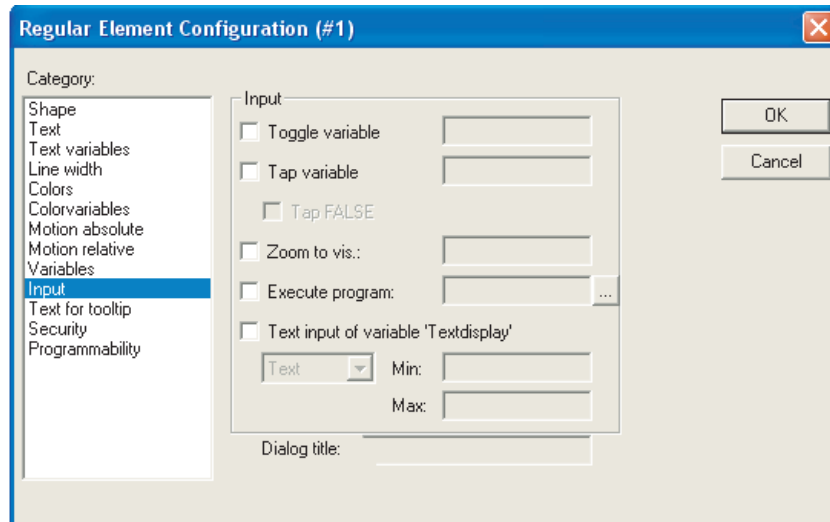
Figure 34: *Finestra di configurazione dell'elemento visualizzato (Categoria Input)*

fase dichiarativa), allora il valore di default è FALSE se l'opzione **Tap False** è disattivata, altrimenti è TRUE.

Zoom to Vis..: Se l'opzione viene attivata è possibile dare in ingresso il nome di una visualizzazione che verrà aperta nel momento in cui si cliccherà sull'elemento in modalità on line. In ingresso è possibile fornire:

- il nome di un oggetto visualizzazione facente parte del progetto corrente.
- un puntatore a oggetto-visualizzazione(instance). Se questa istanza a cui si vuole 'saltare' contiene dei placeholders (vedi paragrafo 3.4.1 "La funzione dei Placeholders"), allora questi possono essere sostituiti da nomi di variabili. Per fare ciò è necessario utilizzare la seguente sintassi:

<nomeVisualizzazione>(<Placeholder1>:=<Testo1>, <Placeholder2>:=<Testo2>, ..
<PlaceholderN>:=<TestoN>
Esempio:
Chiamando la visualizzazione visu1, nella quale erano stati usati i placeholders varRef1 e varRef2, questi vengono rimpiazzati rispettivamente dalle variabili PROG.var1 e PROG.var2
:
visu1(varRef1:=PROG.var1, varRef2:=PROG.var2)

- Se viene digitato il comando "ZOOMTOCALLER", quando si clicca sull'elemento in modalità on line si torna alla eventuale visualizzazione chiamante.

Execute program: Se questa opzione viene attivata è possibile inserire nel campo di testo dei speciali assegnamenti a variabili già dichiarate (ASSIGN) o definire dei comandi interni che verranno eseguiti non appena l'elemento verrà cliccato in modalità on line. Premendo il tasto "**...**" si accede alla finestra **Configure programs** dove si possono selezionare i comandi da eseguire (Add) e disporli nell'ordine desiderato (Before, After).

Text input of variable ’Testdisplay’: Se viene selezionato, allora in modalità on line si ha la possibilità di inserire un valore che dopo aver premuto <**Enter**> verrà scritto sulla variabile che appare nel campo **Textdisplay** della categoria **’Variables’**. Attraverso lo scroll box viene scelto il tipo di dato che è possibile ricevere in ingresso. Selezionando **Text** verrà mostrato in on line mode un campo di testo su cui sarà possibile digitare il valore desiderato. Selezionando invece **Numpad** o **Keypad** verrà visualizzata una tastiera rispettivamente numerica o alfabetica, dove si potrà premere direttamente il valore tra quelli disponibili. Il range valido di questa tastiera viene definito digitando i valori di massimo e di minimo nei rispettivi campi **Max:** e **Min:**

4.3.2 Text for Tooltip

Questa categoria offre la possibilità di visualizzare in modalità on line un testo nel momento in cui il cursore del mouse viene posizionato sopra l’elemento. Il testo che si vuole visualizzare deve essere scritto nell’apposito campo **Content**.

4.3.3 Security

Questa categoria è utile se chi crea l’elemento vuole garantire delle differenti possibilità di accesso e di visualizzazione ai vari user group utilizzanti Codesys definiti precedentemente. Infatti attraverso il comando **’Project’User Group Passwords’** si accede ad una finestra in cui è possibile creare un proprio user-group (contrassegnato da un livello) protetto da Password. Le
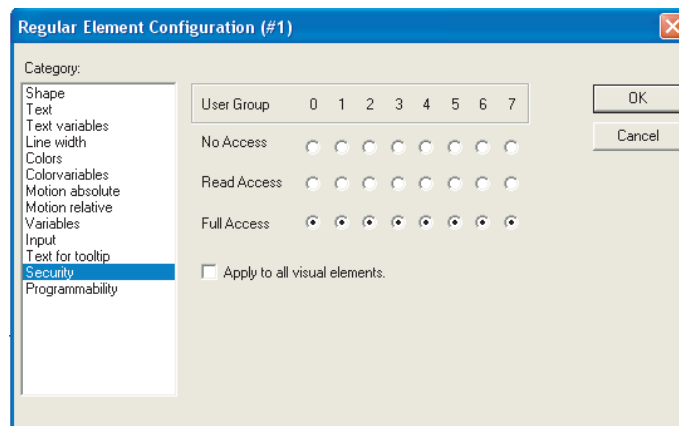
Figure 35: *Finestra di configurazione dell’elemento visualizzato (Categoria Security)*

possibilità di accesso e di visualizzazione che possono essere assegnate all’elemento sono mostrate nella seguene tabella: Se si vogliono assegnare le medesime regole d’accesso anche agli altri el-

No Access	L’elemento non sarà visibile
Read Access	L’elemento sarà visibile ma non utilizzabile (non sono permessi input)
Full Access	L’elemento sarà visibile e utilizzabile (può ricevere ingressi)

ementi presenti nella stessa visualizzazione, basterà attivare l’opzione **Apply to all visual elements**.
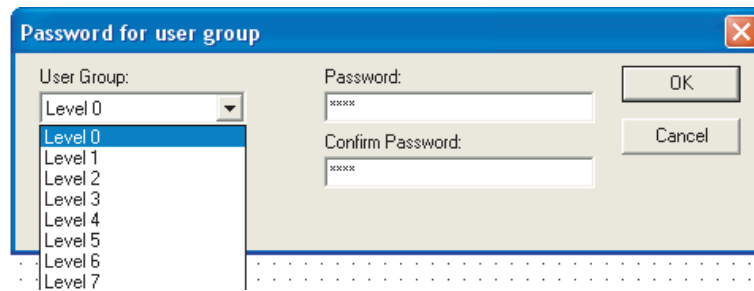
Figure 36: *Finestra di definizione della password di user-group*

4.3.4 Programmability

Le proprietà di ogni elemento che compone una visualizzazione possono essere definite anche attraverso i componenti di una particolare variabile oggetto (di tipo VisualObjectType), che può essere utilizzata esclusivamente per questo tipo di programmazione. Ciò è possibile utilizzando
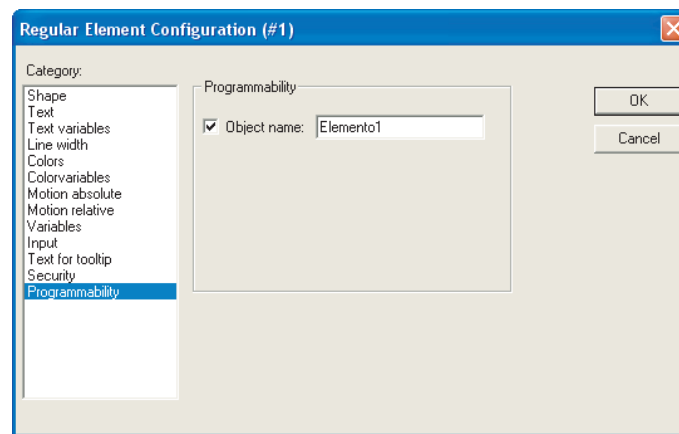
Figure 37: Finestra di configurazione dell'elemento visualizzato (Categoria Programmability)

la struttura **VisualObjectType** che fa parte della libreria **SysLibVisu.lib**. Questa libreria di solito non è presente nella versione demo gratuita di CoDeSys. Per creare una variabile VisualObjectType riferita all'elemento che si sta configurando sarà necessario inserire nel campo di testo una variabile che verrà cosi dichiarata automaticamente. La dichiarazione avverrà in modo implicito e non visibile per l'utente ma la variabile diventerà automaticamente visibile da tutti i programmi del progetto. Per maggiori informazioni sulla struttura VisualObjectType e sull'uso dei suoi vari componenti si consiglia di consultare la tabella di pagina 24 e seguenti del manuale "The Codesys Visualization.

4.4 Bitmap

E' possibile inserire nella visualizzazione corrente anche un un immagine Bitmap attraverso il comando **'Insert"Bitmap'** o usando direttamente il relativo tasto presente nella barra applicazioni dell'Editor grafico. Per configurare questo tipo di elemento sarà necessario digitare la path del file bitmap che si vuole inserire nel campo di testo **Bitmap** presente nella finestra di
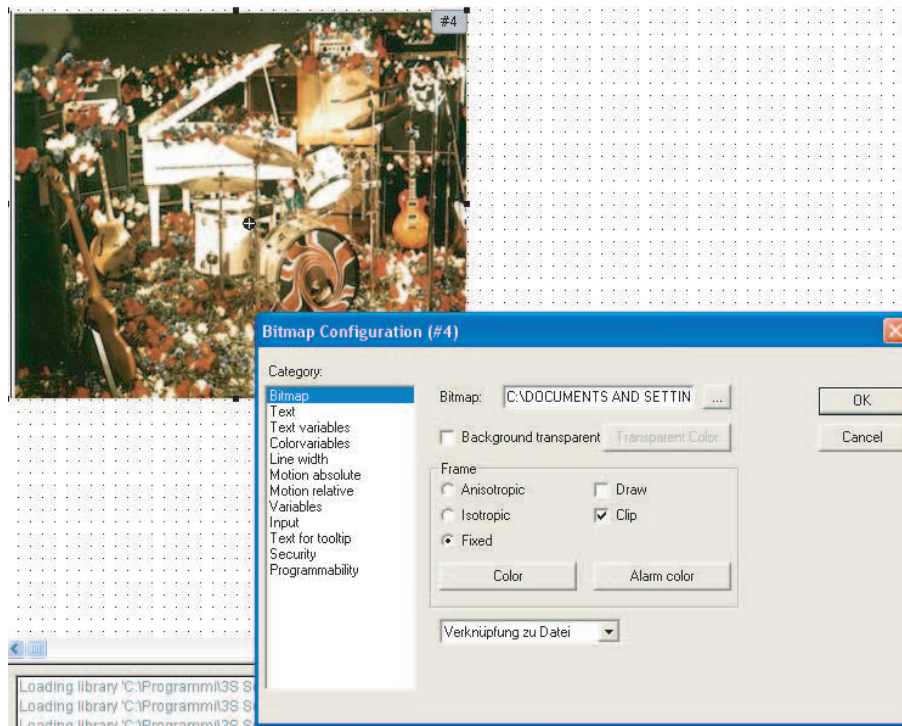
Figure 38: *Finestra di configurazione di immagine Bitmap*

configurazione. Possono essere selezionati inoltre altri parametri utili per per la definizione delle dimensioni e dell'aspetto dell'immagine in modalità on line.

4.5 Visualization

E' possibile inserire all'interno della visualizzazione corrente anche un elemento-visualizzazione, creando una istanza (un riferimento) della visualizzazione che si vuole aprire sopra quella corrente. La configurazione di questa istanza viene definita nella categoria **Visualization** presente all'interno della finestra di configurazione di questo particolare tipo di elemento. Si può utilizzare il tasto **...** per aprire una finestra contenente la lista delle visualizzazioni disponbili all'interno del progetto.

Per spiegare a cosa serva il tasto **Placeholder** presente all'interno di questa finestra di configurazione è necessario descrivere il funzionamento dei placeholders.

4.5.1 La funzione dei Placeholders

Nel momento in cui l'utente va a configurare qualsiasi tipo di elemento ha la possibilità di inserire nei vari campi di testo, un determinato **placeholder** (letteralmente "tieni posto") che rimpiazzi il rispettivo testo o variabile.
Ora, utilizzare questi particolari puntatori ha senso solo se la visualizzazione che si sta configurando non sarà usata direttamente nel programma, bensì se è stata creata per essere inserita come "istanza" all'interno di un altra visualizzazione. Ogni stringa inclusa in due successivi segni a forma di dollaro ($) è un placeholder valido (es: $variable1$).
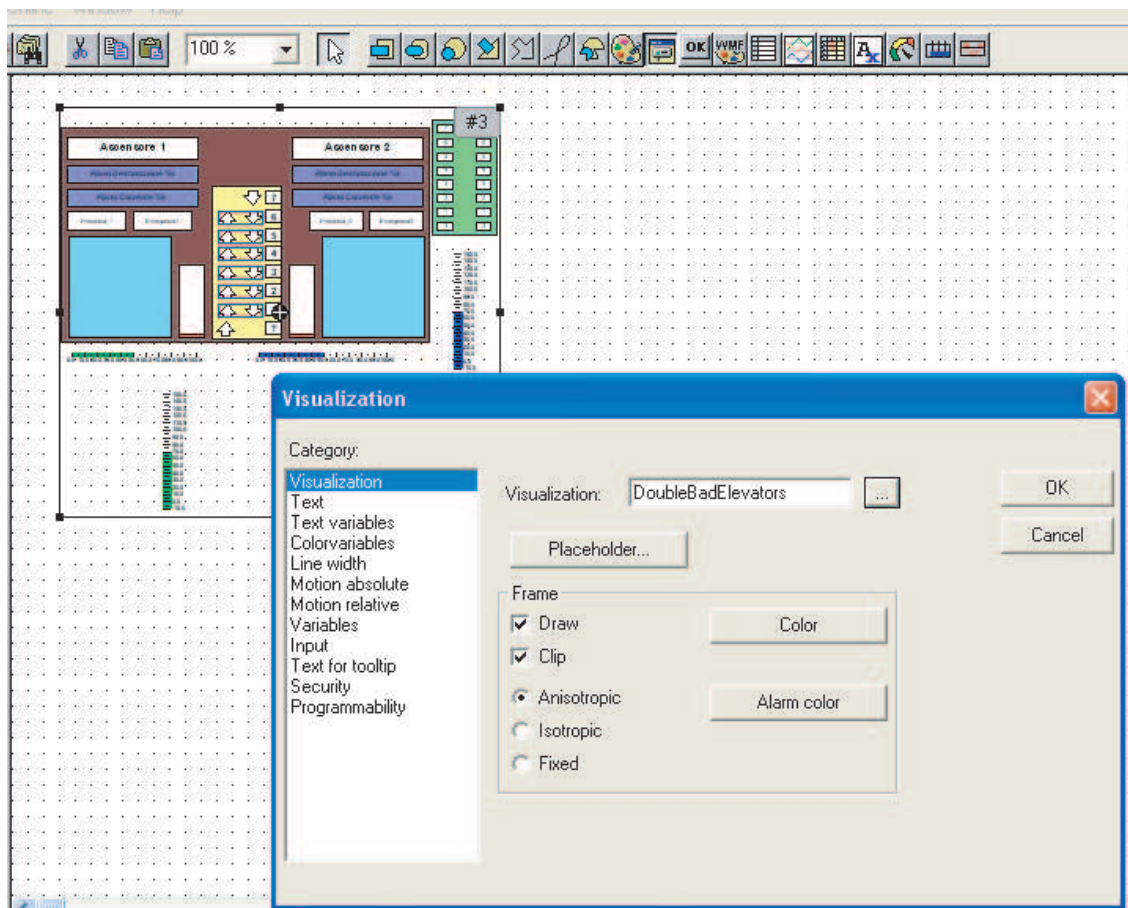
Figure 39: *Finestra di configurazione di un elemento-visualizzazione*

Ogni placeholder che viene usato deve essere inserito in un apposita lista nella finestra **'Extras"List of Placeholders..'**(figura 40) in cui è necessario specificare anche il gruppo di variabili che può essere rimpiazzato da ogni placeholder.
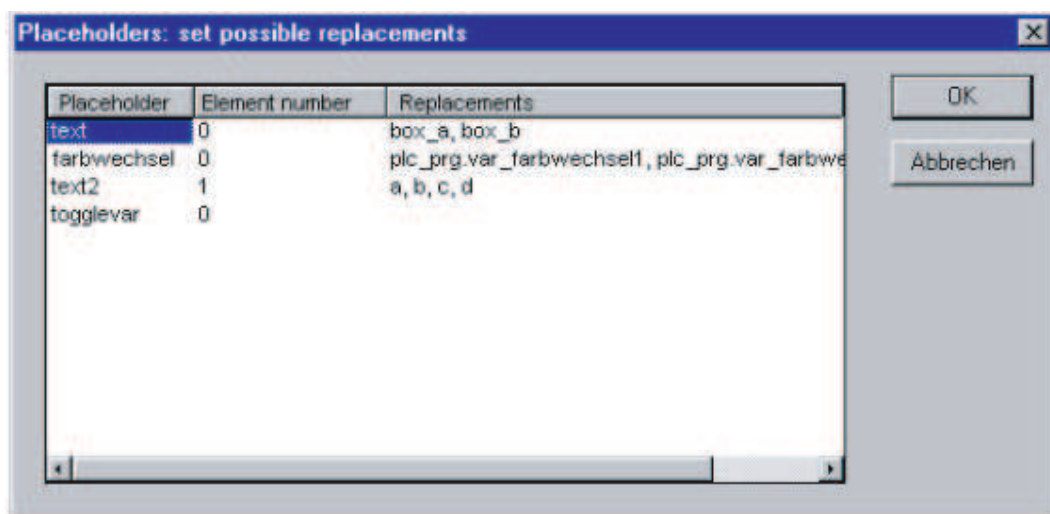
Figure 40: *Finestra* ***'Extras"List of Placeholders..'****. Per ogni placeholder deve esser specificato il numero dell'elemento in cui viene usato e il gruppo di variabili che può rimpiazzare all'interno della definizione una eventuale istanza*

Quando poi si sta definendo nella visualizzazione corrente una istanza di questa visualizzazione, proprio attraverso il pulsante **Placeholder** della finestra di configurazione si accederà alla colonna dei placeholder disponibili e si andrà a definire per l'istanza corrente il valore di ognuno di essi tra quelli che erano stati definiti nella visualizzazione-madre (figura 41).
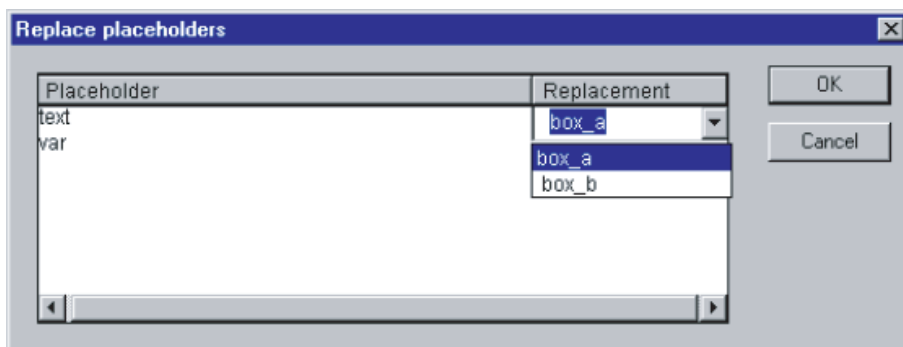
Figure 41: *In questa finestra è possibile definire il valore da attribuire ai placeholder per l'istanza che si sta creando*

4.6 Gruppi di Elementi

Si possono definire gruppi di elementi selezionado i vari componenti tutti in una volta attraverso il mouse. Il gruppo cosi formato potrà essere configurato attraverso il comando **'ExtrasGroup'**,
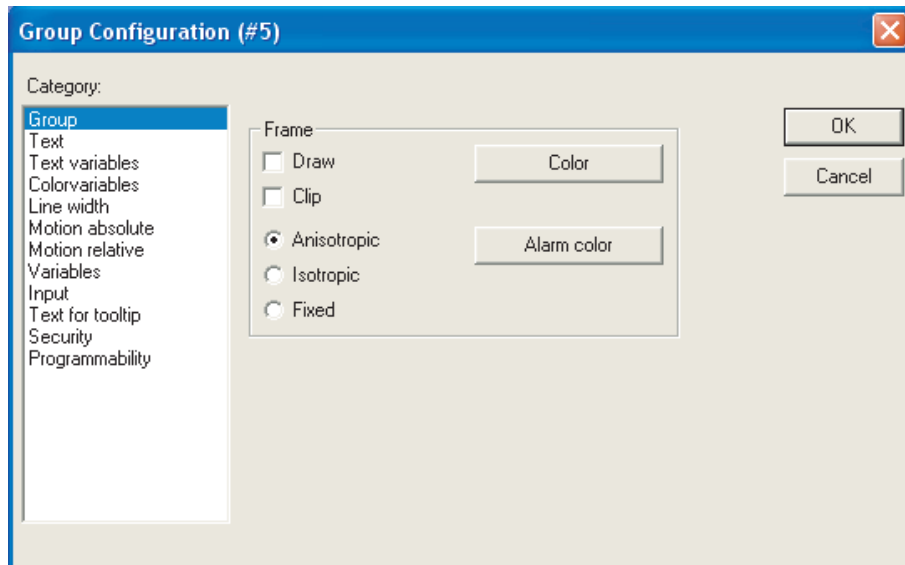
Figure 42: *Finestra di configurazione di un gruppo di elementi*

ogni gruppo si comporterà come un singolo elemento.

5 Esempio

In questa sezione viene proposto un esempio di controllo logico, si vedrà come risolverlo partendo da zero.
L'impianto da controllare, come si può vedere in figura 43, consiste in una cisterna principale
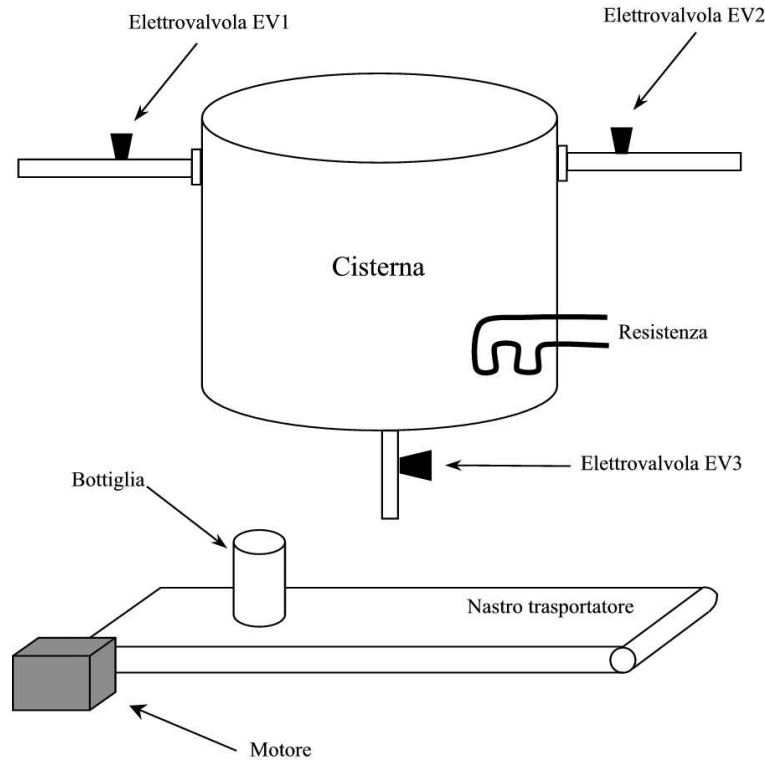
Figure 43: *Impianto di imbottigliamento*

in cui devono essere inseriti due liquidi differenti. Il primo liquido viene immesso nella cisterna tramite l'elettrovalvola EV1 mentre il secondo liquido viene immesso tramite l'elettrovalvola EV2.
Il rapporto in volume tra liquido 1 e liquido 2 deve essere di 4:1. Il livello del liquido all'interno della cisterna è acquisito tramite un sensore analogico. Una volta raggiunto il livello di 200 all'interno del serbatoio si deve permettere ai due liquidi di miscelarsi e di realizzare le opportune reazioni chimiche lasciandoli dentro la cisterna per 10 secondi. A questo punto, terminata la formazione del composto liquido finale, si procede alla sua immissione nelle bottiglie. Quando il liquido è imbottigliato deve essere mantenuto ad una certa temperatura (60°), mediante una opportuna resistenza () che scalda il liquido; le bottiglie sono presenti su un nastro trasportatore attivato dal segnale Motor. Quando il sensore (Pres_Bottle) che segnala che la bottiglia si trova sotto l'elettrovalvola di espulsione del composto è attivo, si può procedere al riempimento della bottiglia (con l'attivazione dell'elettrovalvola EV3). Il riempimento della bottiglia termina quando il suo livello raggiunge il valore di 100. A questo punto il nastro trasportatore può essere riattivato per il riempimento di una nuova bottiglia. Il processo viene avviato dal segnale Start_Proc attivo e può essere interrotto solo quando il serbatoio è completamente vuoto. Nelle

seguenti tabelle sono raccolti tutti i sensori e gli attuatori dell'impianto[1].

Attuatori	**Tipo**	**Descrizione**
Motor	Digitale	Segnale di pilotaggio per il motore che aziona il nastro trasportatore
HotOn	Digitale	Rappresenta il segnale di accensione e spegnimento per scaldare il liquido
EV1	Digitale	Segnale che se vero fa aprire l'elettrovalvola uno
EV2	Digitale	Segnale che se vero fa aprire l'elettrovalvola due
EV3	Digitale	Segnale che se vero fa aprire l'elettrovalvola tre

Table 2: Sono mostrati gli attuatori presenti nell'impianto imbottigliatore.

Sensori	**Tipo**	**Descrizione**
Start_Proc	Digitale	Sensore che rileva l'inizio del processo
Pres_bottle	Digitale	Sensore che rileva la presenza della bottiglia nella zona di riempimento
TemperatureValue	Analogico	Sensore che indica il valore della temperatura in °C
Liv_serb	Analogico	Sensore che indica il livello del liquido contenuto nel serbatoio
Liv_bot	Analogico	Sensore che indica il livello del liquido contenuto nella bottiglia di volta in volta riempita

Table 3: Sono mostrati i sensori presenti nell'impianto imbottigliatore.

Una volta definite le nostre variabili, andiamo ad implementare su Codesys il nostro sistema di controllo logico, ed il programma per la simulazione del Plant, *consideriamo inizialmente il controllo logico senza il controllo della temperatura*. Si crea un nuovo progetto in Codesys, cliccando su `new`, appare una finestra con il nome Target Settings, selezionare `none` nell'opzione `Configuration` (di solito è già di default così) e cliccare su ok. A questo punto appare una finestra come quella di figura figura 44, lasciamo il nome e selezioniamo come tipo di linguaggio SFC, questa POU sarà il programma dove è presente il controllo logico dell'impianto. A questo punto appare la finestra come in figura 45, ci troviamo nell'editor per scrivere un programma in SFC, è possibile vedere come l'ambiente Codesys sia diviso in due parti, sulla sinistra c'è la parte di gestione delle POU, e delle possibili configurazioni del progetto (Risorce, interfaccia grafica per la visualizzaizone, ecc...) mentre nella parte destra è aperto l'editor corrispondente alla parte sinistra selezionata. In questo caso essendo selezionato il programma PLC_PRG è aperto l'editor per scrivere programmi in SFC. A questo punto al fase preliminare di creazione di un nuovo progetto è conclusa. Andando nel browser di progetto nella parte `Resources`, dovrebbero essere visualizzate le librerie inserite di default; Standard e IECSFC come in figura 46 Se il progetto non presenta queste librerie, si deve accedere al menù `Library Manager`, con un doppio click del

[1]I nomi dei sensori e attuatori sono quelli utilizzati nel codice di controllo in CoDeSys.
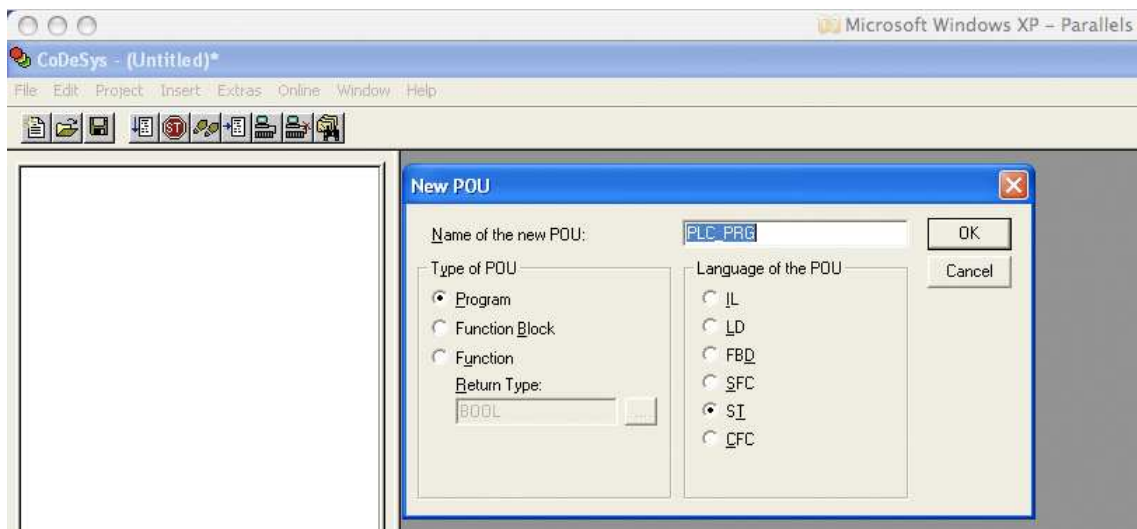
Figure 44: Definizione nuova POU
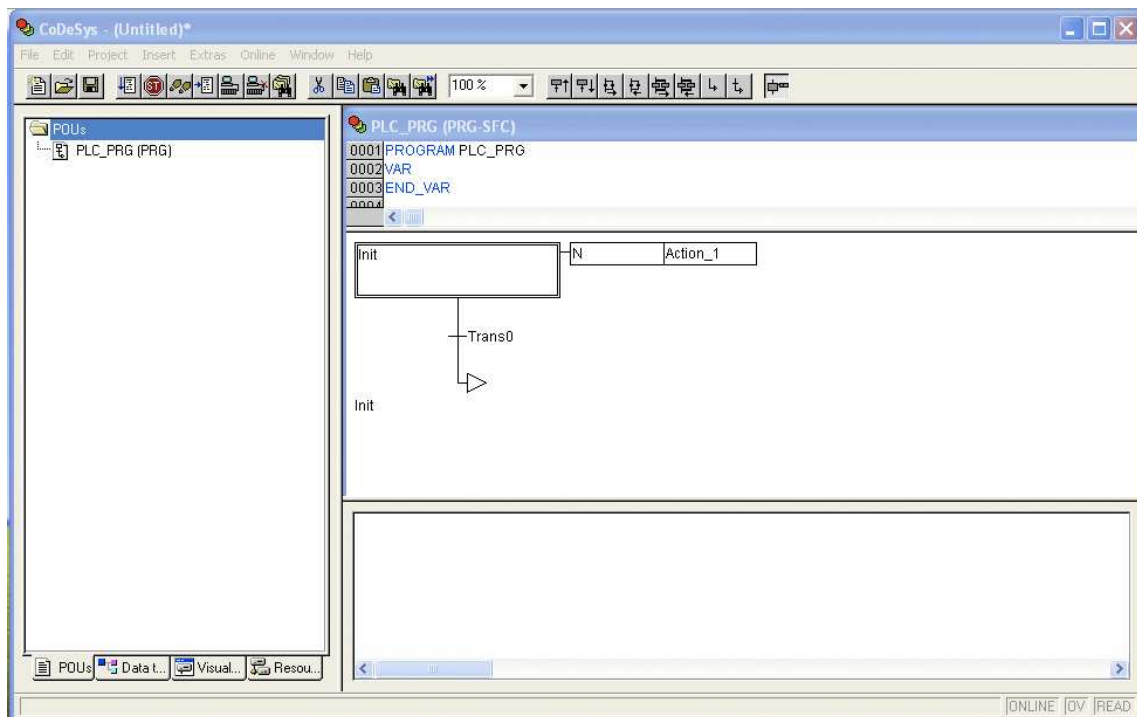
Figure 45: Editor SFC per PLC_PRG
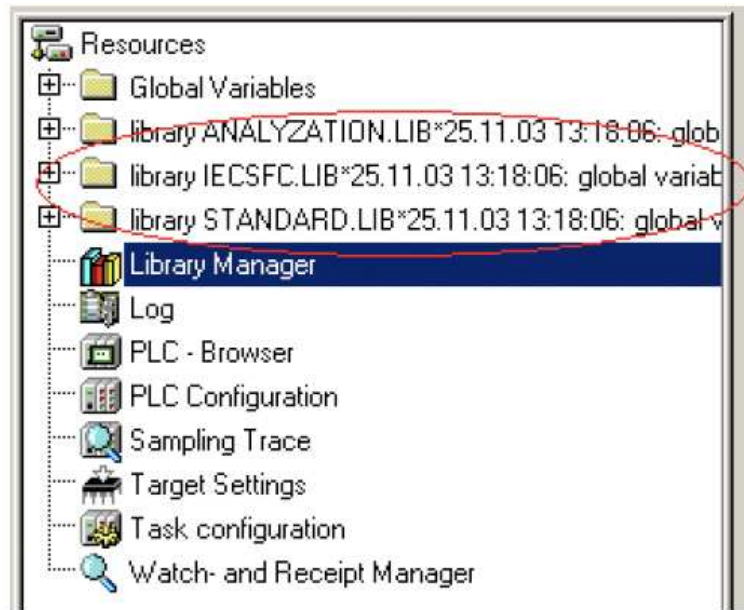
Figure 46: Librerie standard in Codesys

mouse, che farà comparire la schermata come in figura 47. A questo punto cliccando con il pulsante destro nella finestra in mezzo allo schermo con il nome di `Library Manager` e cliccando sul comando `Additional Library` è possibile inserire le librerie standard e IECSFC, quest'ultima serve per la gestione degli step IEC. Per inserire le librerie bisogna selezionare i file con estensione
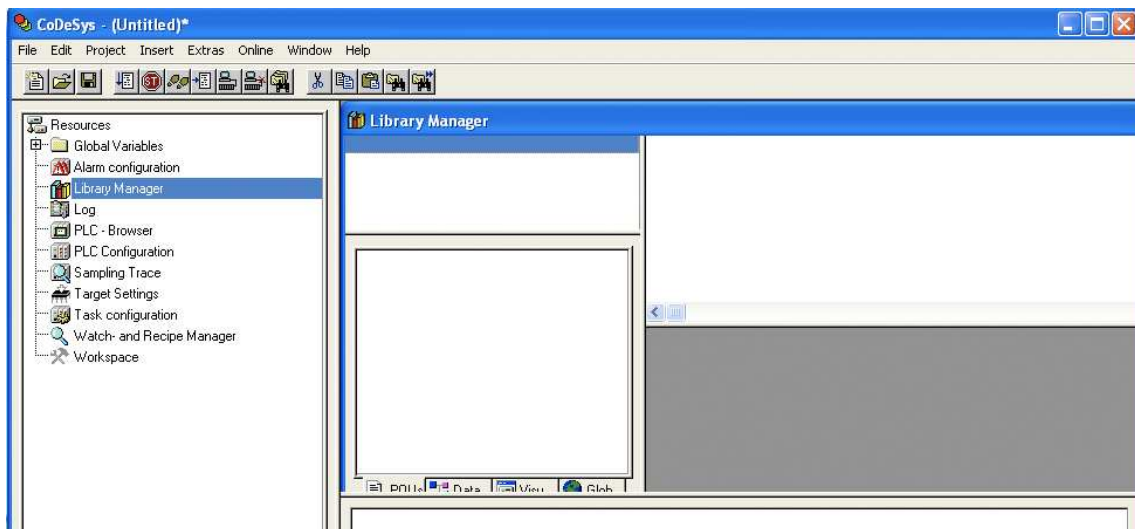
Figure 47: Inserimento di librerie standard in Codesys

.LIB, una volta selezionate il browser di libreria presenta l'elenco delle POU preprogrammate che si possono inserire nel programma in corso di progetto (es. Timer, contatori, ecc.) con sulla destra la loro dichiarazione di variabili e l'aspetto grafico (visibile durante la programmazione nei linguaggi LD e FBD).

Tornando all'editor SFC è possibile ora scrivere il codice relativo al nostro sistema di controllo, come si può notare la barra degli strumenti con l'editor SFC aperto visualizza le possibili modifiche al diagramma che si possono fare:

Step transistion before: inserisce una sequenza passo transizione prima del punto selezionato.

Step transition after: inserisce una sequenza passo-transizione dopo il punto selezionato.

Alternative Branch (right) : inserisce una diramazione alternativa (due transizioni in disgiunzione tra loro), aggiungendo una transizione alla destra di quella selezionata.

Alternative Branch (left): inserisce una diramazione alternativa (due transizioni in disgiunzione tra loro), aggiungendo una transizione alla sinistra di quella selezionata.

Parallel Branch (right): inserisce una diramazione parallela (due sequenze di passi eseguite contemporaneamente), aggiungendo un passo alla destra di quello selezionato.

Parallel Branch (left): inserisce una diramazione parallela (due sequenze di passi eseguite contemporaneamente), aggiungendo un passo alla sinistra di quello selezionato.

Jump: inserisce un salto verso uno stato.

Transition Jump: inserisce un salto e una transizione.

E' possibile ora scrivere il codice relativo al controllo, per inserire le variabili ci sono due possibilità, la prima è quella di scrivere tutte le variabili prima di iniziare il codice, la seconda è quella di inserire la variabili di volta in volta. Infatti mentre si scrive del codice in Codesys se non si scrive una parola chiave (comandi predefiniti) appare una finestra come quella riportata in figura 48, in cui è possibile definire le variabili. All'interno della finestra è possibile definire la classe, (globale, ingresso, uscita, locale ecc...) il nome, il tipo (intera, reale, ecc...) valore iniziale e un commento per descrivere la variabile; per definire una variabile globale, bisogna definirla di classe `VAR_GLOBAL`. E' comunque possibile definire tutte le variabili all'inizio, se una variabile deve avere visibilità solo all'interno di quella POU, la variabile va definita all'interno della POU stessa nella parte superiore dell'editor, se invece la variabile deve essere visibile a più POU allora questa va definita nell'editor delle variabili globali, che si trova nel menù `Resources` alla voce `Global_Variables`.
Quando prima abbiamo definito le variabili di ingresso ed uscita per il programma di controllo, le abbiamo definite rispetto all'implementazione su un PLC. In modalità di simulazione però le variabili di ingresso ed uscita, devono poter essere manipolate anche dal programma per la simulazione del sistema, quindi queste variabili devono avere una visibilità non locale ma globale. Tutte le variabili che abbiamo definito precedentemente come di ingresso ed uscita, andranno dichiarate globali perchè vengono utilizzate anche dal programma per la simulazione del plant.
Una volta ultimato il programma in SFC è possibile scrivere il programma per la simulazione del plant, per definire una nuova POU basta cliccare con il tasto destro sull'icona `POUs` e selezionare il comando `Add object`, a questo punto compare una finestra come quella iniziale (figura 44) da qui è possibile definire una nuova POU scegliendo il linguaggio desiderato. Ad esempio per la scrittura del programma per la simulazione del plant è possibile usare il linguaggio ST perchè rende più semplice l'implementazione del programma. Terminata la scrittura del programma
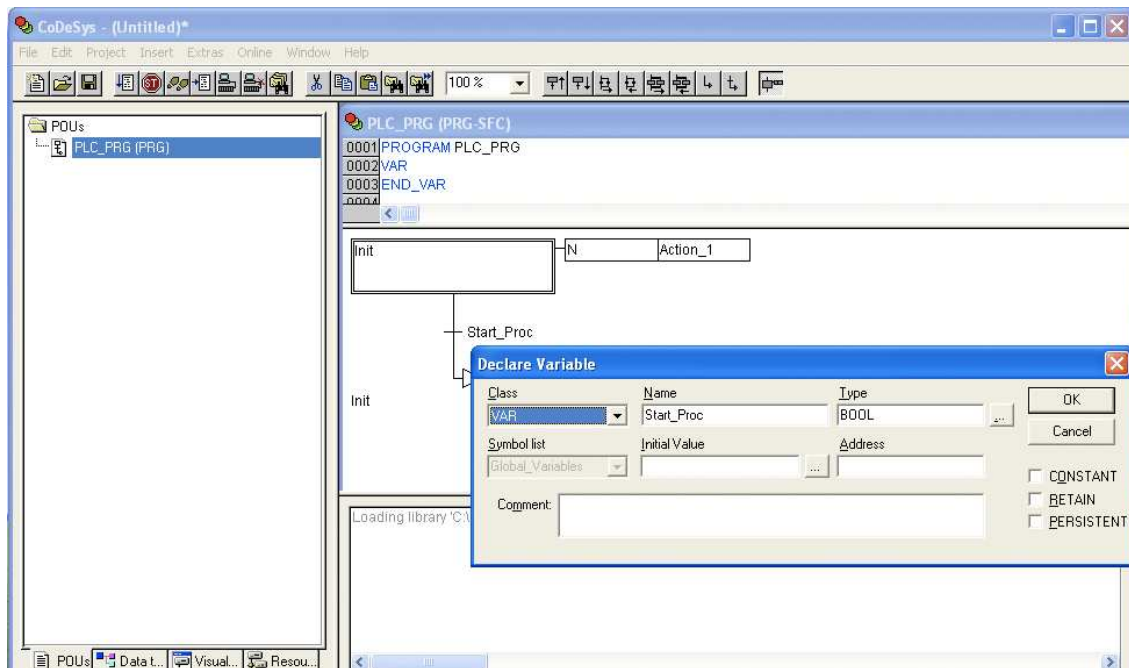
Figure 48: Inserimento delle variabili.

per la simulazione del plant, è possibile definire una interfaccia grafica tramite la quale è possibile interagire con i programmi. Selezionando nel browser di progetto di Codesys nella parte `Visualization` è possibile inserire degli oggetti, cliccando con il tasto destro su `Visualization` ed usando il comando `Add object`, nel quale è possibile inserire degli elementi grafici come forme geometriche, immagini ed oggetti predefiniti per la visualizzazione di valori (istogrammi, display virtuali, ecc...) e pulsanti da collegare alle variabili. Ciascuno di questi oggetti può essere inserito selezionando il pulsante relativo nella barra degli strumenti in lato nella schermata, puntando il mouse in un punto del foglio e trascinando il puntatore tenendo premuto il tasto sinistro per impostare la grandezza dell'oggetto. Per una descrizione approfondita degli oggetti grafici guardare la sezione 4.
A questo punto una volta completata la scrittura di tutte le componenti del progetto, bisogna inserire le POU desiderate nel ciclo di esecuzione (come spiegato nella sezione 3.6), nel nostro caso andando nel menù `Resources` e scegliendo `Task Configuration` dobbiamo andare ad inserire uno o più task, a seconda di come vogliamo implementare le nostre POU. Una volta completato un progetto nel suo insieme (programmi, definizioni delle variabili) è possibile verificarne la correttezza selezionando il comando `Build` dal menù `Project`. Se il comando non è corretto, verrà visualizzato l'errore tra i messaggi tra i messaggi nella parte in basso a destra della finestra Codesys; cliccando due volte sull'errore selezionato, Codesys apre automaticamente il programma nel punto dell'errore.
Se il progetto è corretto è possibile eseguirlo in modalità simulazione, attraverso il menù `online` ed il comando `login`, ed una volta effettuato il login è necessario lanciare l'esecuzione del programma mediante il comando `Run`. A questo punto il programma è in esecuzione sul PC, il quale simula il funzionamento del PLC in modo virtuale. nel menù online deve essere impostata l'opzione `simulation mode`, se all'inizio del progetto si è impostato il target su `None` questa
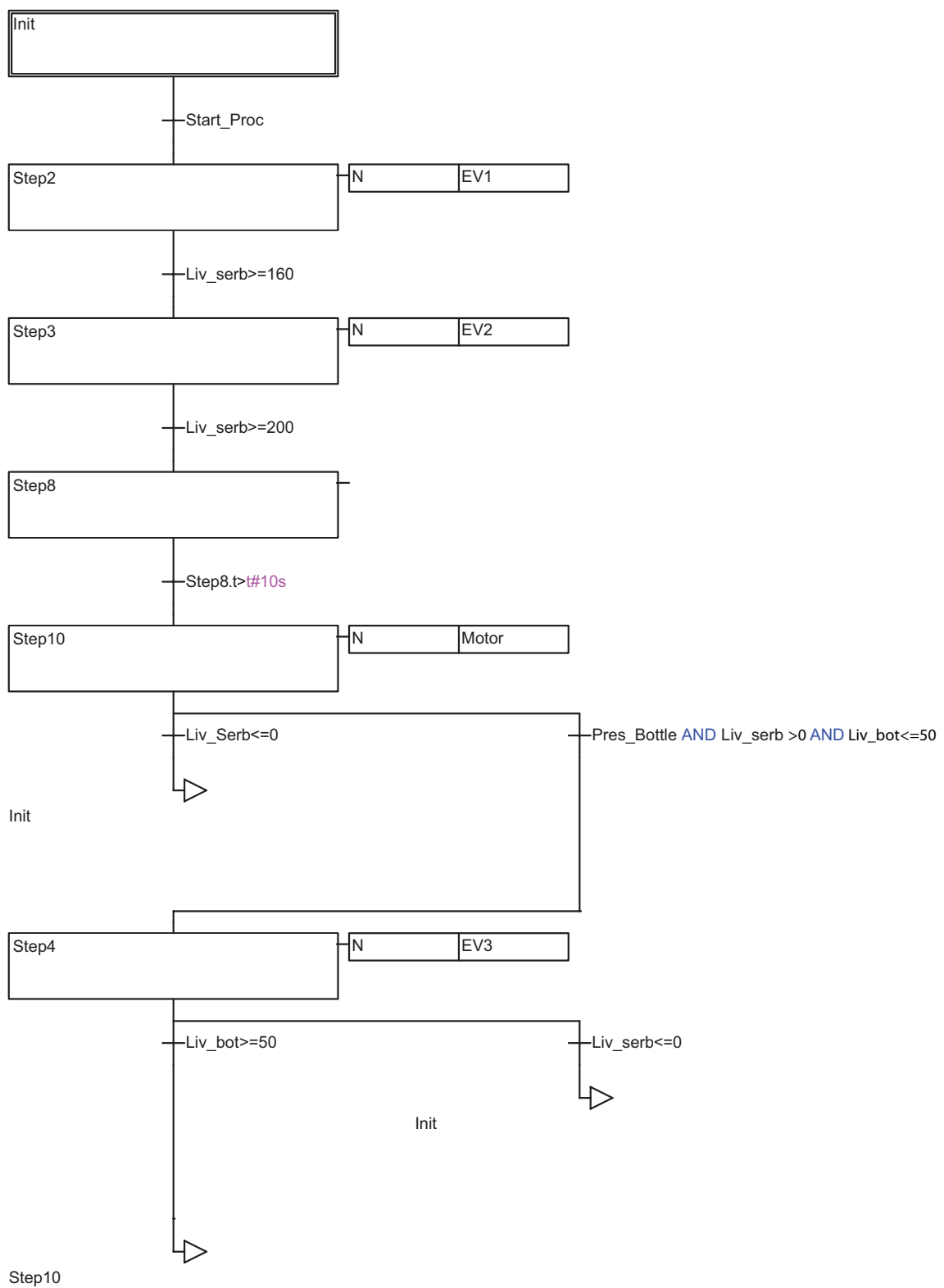
Figure 49: SFC del programma di controllo logico.

```
0001
0002 (*Riempimento del serbatoio*)
0003 IF (EV1 OR EV2) THEN
0004       Liv_serb:=Liv_serb+1;
0005 END_IF;
0006
0007 (*Svuotamento serbatoio riempimento bottiglia*)
0008 IF (EV3 AND Liv_Serb>0 )THEN
0009       Liv_bot:=Liv_bot+1;
0010       Liv_serb:=Liv_serb-1;
0011       IF Liv_serb<=0 THEN
0012             Liv_serb:=0;
0013       END_IF
0014 END_IF;
0015
0016 IF Motor THEN
0017       Pos_bottle:=Pos_bottle+5;
0018 END_IF;
0019
0020 IF (Pos_bottle>=195 AND Pos_bottle<=205) THEN
0021       Pres_bottle:=TRUE;
0022 ELSE
0023       Pres_bottle:=FALSE;
0024 END_IF;
0025
0026 IF Pos_bottle>=300 THEN
0027       Pos_bottle:=0;
0028       Liv_bot:=0;
0029 END_IF;
```

Figure 50: ST del programma di simulazione plant.

opzione dovrebbe essere già impostata e non modificabile.
Viene riportato in seguito il codice SFC della logica di controllo, ed il codice ST per il programma per la simulazione dell'impianto.

6 Esempio con attuatore generalizzato

6.1 Il concetto dell'attuatore generalizzato

Nella scrittura dei software c'è stata una evoluzione, con l'aumentare della complessità si sono dovute trovare tecniche che andavano oltre al concetto classico di funzione, e così è nata l'idea di *incapsulamento* cioè del creare degli oggetti base che potevano essere riutilizzati in più software. Questa teoria della programmazione prende il nome di *programmazione ad oggetti*.
Per spiegare meglio come tutto questo discorso si possa abbinare anche al controllo di sequenze si può pensare al semplice esempio di un impianto che, in un particolare momento, deve aprire una porta. Si possono avere tre tipi di sensore che mi dicono in che stato è la porta:

A Singola Retroazione Un sensore di questo tipo mi dice quando l'azione è finita, nel nostro caso avrà un valore `TRUE` quando la porta è completamente aperta o `FALSE` quando è chiusa o è in fase di movimento.

A Doppia Retroazione Questo tipo mi darà informazione sia sulla partenza che sulla conclusione dell'operazione. Con una prima variabile mi dirà se la porta è completamente chiusa o no, mentre con una seconda variabile se è completamente aperta o no. Combinando le due informazioni posso sempre sapere in che condizione si trova il mio sistema.

A Time-Out Questo non è un sensore vero e proprio. Io so quanto tempo ci vuole per fare una operazione quindi immagino che, passata quella quantità di tempo dopo l'emissione del comando, quella certa operazione sia effetivamente conclusa, tenendo conto magari di un certo margine di errore.

In ambito controllistico l'astrazione si ottiene dividendo la politica di gestione dal meccanismo di funzionamento. Avrò cioè un gestore che mi da il comando di aprire la porta e un oggetto, che è l'insieme di uno o più attuatori e uno o più sensori, che mi esegue l'operazione voluta azionando un certo meccanismo. Questo oggetto di controllo è l'***Attuatore Generalizzato***. L'astrazione si vede se si pensa al fatto che al gestore non interessa che tipo di sensore usa l'attuatore generalizzato per aprire la porta. L'importante è che quando arriva il comando di apertura la porta si apra. Oltre all'astrazione si possono ritrovare le proprietà di incapsulamento e riutilizzabilità, infatti il meccanismo di funzionamento è incapsulato all'interno dell'oggetto attuatore generalizzato che può essere riutilizzato tranquillamente in un impianto completamente diverso ma che ha una operazione simile all'apertura della porta.
Possiamo classificare questi oggetti in due famiglie di attuatori generalizzati:

Do/Done Questo tipo di attuatore viene fatto partire dal controllo principale con un comando "Do". L'apertura della porta è proprio un esempio di questo tipo di attuatore generalizzato.

Start/Stop In questo caso è il controllo principale che fa partire e ferma l'oggetto, usando due comandi (tipicamente "Sart" e "Stop"). Per esempio il controllo di temperatura di un

liquido viene fatto partire e continua fintanto che il controllo principale dice che non è più necessario.

6.2 Imbottigliatore con attuatore generalizzato

Supponiamo ora di voler aggiungere al nostro impianto la funzione per il mantenimento ad una certa temperatura del liquido miscelato durante il processo di imbottigliamento. Si avrà un sensore TemperatureValue che indicherà il valore della temperatura del liquido ed una resistenza comandata attraverso il comando HotOn.
Aggiungere questa funzione al nostro impianto dato il modo in cui è stato scritto il codice precedentemente non è così agevole vista la sua scarsa modularità, lo stesso vale ad esempio se cambiassimo dei sensori. Ad esempio invece di avere un sensore di livello della bottiglia, vogliamo mettere un sensore di peso per sapere quando la bottiglia è piena, la logica dell'impianto non cambia, cambia solo l'implementazione. Anche questa variazione nel codice non sarebbe agevole, visto che sono state completamente mischiate le funzioni da implementare con i meccanismi con cui si vogliono impelmentare. Analizziamo ora il nostro processo e scomponiamolo nelle funzioni base che il nostro processo esegue per ottenere la funzione complessiva.
Il primo passo da fare per la definizione degli attuatori generalizzati è quello di indetificare le funzionalità del sistema, cioè delle funzioni/operazioni di base che devono essere eseguite dal sistema.
Nel sistema di esempio se ne possono individuare quattro:

1. Riempimento del serbatoio.
2. Posizionamento delle bottiglie.
3. Svuotamento del serbatoio.
4. Controllo della temperatura.

La prima ha solo il compito di immettere i due liquidi nella cisterna nel rapporto indicato (per generalizzare il tutto si può anche permettere un rapporto variabile) e contare i 10 secondi necessari per il miscelamento.
La seconda gestirà il nastro trasportatore per mettere in posizione corretta le bottiglie. Qui si inizia già a notare la riutilizzabilità: se abbiamo una altro sistema che ha una funzionalità che prevede il posizionamento, possiamo benissimo utilizzare l'attuatore generalizzato che creremo per l'imbottigliatore, basta solo cambiare i parametri che dicono al nostro oggetto quale motore e quale sensore usare per la sua funzionalità, senza dover riprogettare tutto dall'inizio.
La terza funzionalità si occuperà di svuotare lentamente il serbatoio fino al riempimento di una bottiglia.
Infine la quarta userà la resistenza per scaldare la miscela ed effettuare una isteresi della temperatura.

6.3 Descrizione della sequenza

Una volta che sono state trovate le funzionalità di base bisogna indicare la sequenza con la quale vengono usate. Questa descrizione serve per capire come e quando vanno usati i vari attuatori generalizzati creati quando si andrà a implementare il controllo principale. Nell'esempio

dell'impianto di imbottigliamento la sequenza è la seguente:

```
Riempimento Serbatoio- Posizionamento - Riempimento Bottiglia - Posizionamento
- Riempimento Bottiglia - Posizionamento - Riempimento Bottiglia - ....
```

fino a quando il serbatoio è completamente vuoto, nel qual caso si riparte con un altro `Riempimento Serbatoio`. In parallelo a tutto questo c'è il `Controllo Temperatura` solo quando c'è del liquido all'interno della cisterna.
Dopo che sono state definite le funzionalità e la sequenza con la quale sono usate bisogna andare ad un livello più basso ed iniziare a pensare a quali sensori e attuatori servono per per permettere all'attuatore generalizzato di eseguire la sua funzionalità. E' anche possibile che siano già stati definiti prima i sensori e gli attuatori da utilizzare, in questo caso basta semplicemente associarli all'attuatore generalizzato. Nell'impianto di imbottigliamento delll' esempio i sensori ed attuatori erano già assegnati:
Sensori:

- 2 sensori analogici di livello, uno per la cisterna e uno per la bottiglia;
- 1 sensore digitale di presenza per le bottiglia;
- 1 sensore analogico di temperatura per la cisterna.

Attuatori:

- 3 elettrovalvole di tipo on/off per l'immissione dei liquidi e l'espulsione della miscela;
- 1 motore per la movimentazione del nastro trasportatore;
- 1 resistenza per il controllo della temperatura del liquido all'interno della cisterna.

6.4 Accoppiamento sensore/attuatore

Nella sezione precedente è stata riportata la lista dei sensori e degli attuatori, ora dobbiamo legare questi sensori ed attuatori alle funzionalità per definire gli attuatori generalizzati che useremo. Infatti l'Attuatore Generalizzato realizza la sua funzione usando un certo numero di sensori e attuatori reali, che vengono "nascosti" dentro la funzione stessa. Tutto questo serve per separare la politica di gestione dal meccanismo di funzionamento. Infatti il meccanismo è implementato all'interno degli attuatori generalizzati, mentre la politica di gestione dell'impianto è nel programma principale, il quale non vede i singoli sensori o i singoli attuatori ma dialoga solamente con gli Attuatori Generalizzati. Questa separazione politca/meccanismo è la base della riutilizzabilità. Per migliorare la leggibilità e la funzionalità del sistema bisogna cercare di creare degli oggetti disgiunti, cioè ogni sensore e ogni attuatore devono essere associati ad un solo attuatore generalizzato.
Ritornando al nostro esempio, questo accoppiamento risulta essere molto semplice, grazie alla semplicità del sistema stesso.

1. Riempimento serbatoio:

- sensore analogico per il livello della cisterna
- 2 elettrovalvole di tipo on/off per l'immissione dei due liquidi

2. Posizionamento bottiglia:
    - sensore digitale di presenza bottiglia
    - motore per la movimentazione del nastro trasportatore
3. Riempimento Bottiglia:
    - sensore analogico per il livello della bottiglia
    - elettrovalvola di tipo on/off per l'espulsione della miscela
4. Controllo temperatura:
    - sensore di temperatura
    - resistenza.

Con questo punto è terminata la parte di progettazione del controllo del sistema, ora rimane l'implementazione pratica.

6.5 Implementazione in Codesys

Per implementare l'attuatore generalizzato useremo i function block. Andiamo a definire il function block per il riempimento del serbatoio, come mostrato in figura 51. Nella parte sinistra della finestra si definisce il tipo POU, nel nostro caso Function Block, mentre nella parte destra si definisce il linguaggio con il quale si vuole definire la POU; un Function Block può essere definito con qualsiasi tipo di linguaggio. nella finestra dove sono l'elenco delle POU è possibile vedere che le POU che sono function block hanno tra parentesi al sigla (FB). Una volta che abbiamo definito Il FB, è come se avessimo definito un tipo di dato, definendo il FB per il serbatoio non abbiamo definito il programma per implementare l'attuatore generalizzato per il controllo del livello del serbatoio, ma abbiamo definito il tipo di dato con il quale potremo definire l'istanza per l'attuatore generalizzato.
Per definire l'attuatore generalizzato, possiamo definire una POU di tipo Program, e come linguaggio della POU il linguaggio FBD (Function Block Diagram), questo ci permette di creare un attuatore generalizzato del tipo di function block che abbiamo definito precdentemente.
Il FBD è definito per righe, su ogni riga è possibile mettere un elemento, oppure una serie di elementi. Quando si definisce il programma è presente una sola riga e su questa ci sono 3 punti interrogativi ed un piccolo box tratteggiato, cliccando su questo con il tasto destro del mouse e cliccando su box è possibile inserire il blocco desiderato. Apparirà di default il block AND, cliccando sulla scritta AND e premendo il tasto F2 (Input Assistant) è possibile richiamare il function block creato in precedenze. Premendo F2 appare una finestra (come quella di figura 16) con tutti i tipi di function block, cliccando su `User defined Function Block` è possibile richiamare i FB definiti dall'utente. Se ad esempio avevamo chiamato il nostro FB come LevelFB, apparirà un blocco chiamato LevelFB che avrà gli ingressi e le uscite definite nel FB. Ora è possibile assegnare una istanza a quel FB, infatti sopra blocco bisogna mettere il nome di quella
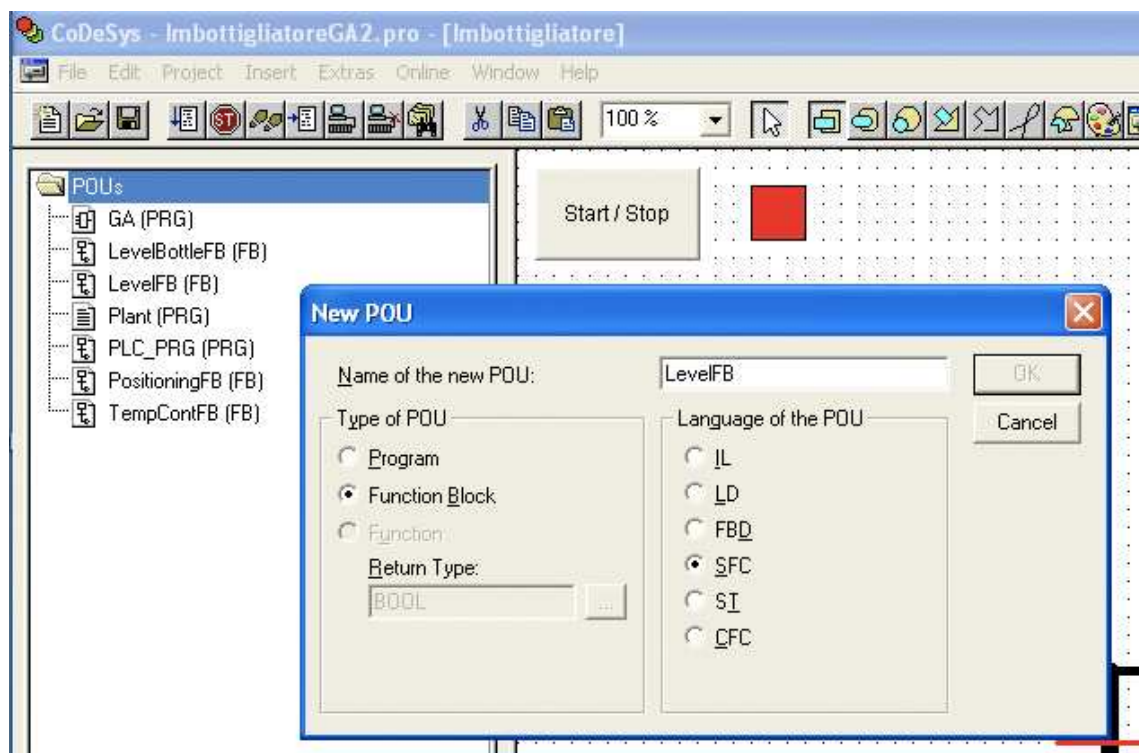
Figure 51: Definizione del function block.

Figure 52: Definizione di un attuatore generalizzato
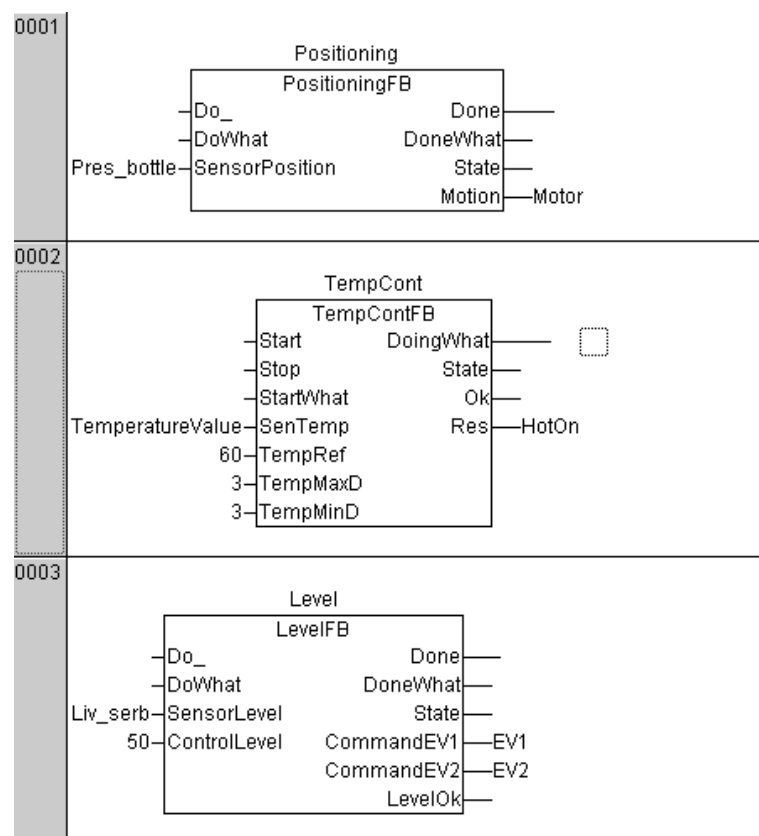
Figure 53: Finestra del function block digram.

istanza, chiamiamola ad esempio TankLevel, una volta finita di scrivere apparirà la finestra per l'assegnamento delle variabili come in figura 52, si vede che LevelFB è un tipo di variabile, mentre TankLevel è un dato di quel tipo di variabile. Per definire un altro GA, bisogna inserire una riga nella finestra, per fare questo bisogna cliccare con il tasto destro e cliccare su `network(before)` oppure `network(after)`, in questo modo si inserisce una riga prima o dopo della riga selezionata. Inserendo più righe è possibile inserire tutti i GA che servono per il nostro controllo logico, come in figura 53.
Una volta definiti tutti i GA bisogna inserire nel task configuration il programma FBD dove sono presenti le definizioni dei vari GA. I GA definiti in questo modo diventano delle entità sempre vive, in questo modo ogni singolo GA può incapsulare la diagnostica locale e lasciare alla politica solo la diagnostica riguardante guasti ad alto livello (come possono essere una errata sequenza di operazioni). La politica dovrà soltanto azionare il GA ed aspettare che il GA finisca la proprio operazione, in questo modo c'è una completa divisione tra politica ed implementazione del meccanismo. Naturalmente ci dovrà essere una gestione di sincronizzazione, per i GA, ad esempio per un GA di tipo Do-Done la politica dovrà dare il segnale Do controllando che il GA sia libero (in stato di Ready) e una volta che ha ricevuto il segnale di Done, dovrà resettare il segnale di Do (in maniera analoga il GA dovrà agire sul segnale di Done). Per la definizione dei singoli GA per l'esempio dell'imbottigliatore si veda l'esempio ImbottigliatoreGA.pro